Mémoire de DEA

DEA Informatique et Ingénierie
Option Informatique des Systèmes de Production

présenté au

Laboratoire d'Informatique

Université Blaise Pascal
Clermont-Ferrand II

par

Jérôme Darmont

Sujet : Comparaison de trois méthodes de groupement d'enregistrements (clustering) pour des bases de données orientées-objet en termes de temps de réponse et d'occupation disque.

University of Oklahoma
School of Computer Science
Norman, Oklahoma

Février-Juin 1994

Abstract

The aim of this study is to compare three clustering methods designed for object-oriented databases (Cactis, CK and ORION) in terms of response time and disk space.

The three algorithms have first been studied in detail, then a comparison methodology for clustering techniques has designed. The actual comparison of the three algorithms has been performed using simulation. Simulation was performed with SLAM II on DECstation 5000/25 workstations.

Simulation experiments we performed showed that Cactis algorithm is better than ORION algorithm and that CK algorithm totally outperforms both other algorithms in terms of response time and clustering overhead.

Keywords: Cactis, CK, Clustering, Object-Oriented Databases, ORION, Simulation, SLAM II

Résumé

Le but de cette étude est la comparaison de trois méthodes de groupement d'enregistrements (clustering) pour des bases de données orientées-objet (Cactis, CK et ORION) en termes de temps de réponse et d'occupation disque.

Après une étude détaillée de chaque algorithme, une méthodologie de comparaison des techniques de clustering a été élaborée, puis appliquée aux trois algorithmes étudiés. Elle utilise la simulation. Les simulations ont été effectuées avec le logiciel SLAM II sur des stations de travail DECstation 5000/25.

Les simulations que nous avons effectuées ont montré que Cactis est un algorithme plus performant qu'ORION, mais que CK est de loin l'algorithme le plus performant en termes de temps de réponse et de surcharge due au clustering.

Mots-clés : Bases de données orientées-objet, Cactis, CK, Clustering, ORION, Simulation, SLAM II

à Anne-Gaëlle

Acknowledgments

I first would like to thank my tutor professor Dr. Le Gruenwald for her kindness and patience while I was working with her.

I also have to thank all the people of the OIP (Office of International Programs) for their warm welcome, and especially Seymour Feiler with who having lunch has always been a great pleasure; along with the Dean of the College of Engineering Dr. Billy Crynes.

Special appreciation to my roommate Jesus Fernando Sotelo who introduced me to the greatest shows on American TV, i.e., NBA basketball plays (Go Bulls!) and "Beavis and Butt-Head" on MTV (Huh huh. This roommate dude was cool!).

I eventually thank Mr. Ammar Attoui and Miss Frédérique Bullat from Blaise Pascal who stayed in contact with me and provided help via E-Mail during this internship.

Table of Contents

Introduction

This research internship has been performed in order for me to obtain the French DEA (qualifying degree for research) in Computer Science. It lasted for five months (from February 1994 to June 1994) and took place at the School of Computer Science of the University of Oklahoma. My tutor professor was Dr. Le Gruenwald.

Database Management Systems (DBMSs) have long been successful in business, but they have not been fully utilized for advanced applications, such as Office Information Systems (OIS) and Computer-Aided Design (CAD). These applications have new requirements in design environments, transactions mechanisms and complex or multimedia types. Object-Oriented Databases (OODBs), integrating techniques from databases, object-oriented languages, programming environments and user interfaces, are built with such advanced applications in mind.

This study deals with clustering in the field of OODBs. There are several ways to improve response time (i.e., to limit the number of disk Input/Output) in a DBMS. Indexing, clustering (i.e., storing related entities close together on secondary storage) and buffering (i.e., fetching clustered entities at the same time and setting up replacement strategies) are widely used techniques in conventional DBMSs. However, OODBs present additional semantics like structural properties (inheritance, composite objects) and interrelationships between objects. New techniques have then to be thought of.

It is widely acknowledged that good object clustering is critical to the performance of OODBs [TSANGARIS91]. Clustering means storing related objects close together on secondary storage so that when one object is accessed from disk, all its related objects are also brought into memory. Then access to these related objects is a main memory access that is much faster than a disk access. Clustering is used to minimize I/O when retrieving a set of related objects.

We have chosen to study three clustering algorithms found in the literature that we consider to be different enough to be representative of the current research on clustering techniques in OODBs (Cactis, CK and ORION clustering algorithms). The Cactis and ORION clustering algorithms are already implemented in DBMSs.

These particular algorithms have been selected because they present characteristics that are interesting to compare. For instance, the CK and ORION are dynamic clustering algorithms as Cactis clustering algorithm is static. ORION also uses only users' hints to cluster a database; the Cactis clustering algorithm uses only statistics about the database and the CK algorithm makes use of both.

Furthermore, the aim of previous performance evaluations performed on these algorithms was only to compare the effects of one particular clustering strategy to those of a "no clustering" policy. We intend to compare each of these three algorithms to each other to determine which one performs the best in a given environment.

The motivation of this research is to find which algorithm is the best, since each of them has until now only been evaluated separately. The characteristics that make this algorithm the best should be isolated. Then this algorithm might be modified to make it even better by investigating its weaknesses and maybe compensating these weaknesses using the other algorithms' strengths.

The first step of this work was to study each clustering algorithm to well understand how it works and to later be able to implement it in a simulation model. At the same time, other papers about various clustering strategies used in OODBs have been collected.

Then comparison criteria have been selected and a comparison methodology has been designed to compare performances of the three algorithms. A literature survey about existing simulation models, performance measurements and performance benchmarks has been gathered at this time.

The actual comparison was done using simulation.

The schedule we adopted is the following:

- study of the three clustering algorithms: one month (February);
- design of a comparison method for the algorithms: one month (March);
- design of the simulation models (conceptual models first then implementation in SLAM II simulation language), simulation results gathering: two months (April and May);
- last simulation results, report and paper writing: one month (June).

The University of Oklahoma

I- The University

The University of Oklahoma (OU) is a major national research university serving the educational, cultural and economic needs of the state, region and nation. OU was created by the Oklahoma Territorial Legislature in 1890, 17 years before Oklahoma became a state. Today the University offers a wide range of academic programs in many different areas. There are 125 degree programs at the undergraduate level, 129 at the master's level, 81 doctoral programs and professional or combined doctoral/professional degrees in 14 areas. OU enrolls more than 24,000 students on campuses in Norman, Oklahoma City and Tulsa and has approximately 1,500 full-time faculty members. The university's annual operating budget is almost $500 million.

The main part of the University is located on the 381 acre (154 ha) Norman Campus. It houses 11 colleges. The North Campus, also in Norman, includes an industrial park and the university airport. The medical and health-related colleges of the university are located on the Health Science Center Campus (HSC) in Oklahoma City.

II- Norman Campus

Today Norman has a population of 79,000 and is thereby the fourth biggest city in Oklahoma after Oklahoma City, Tulsa and Lawton. Norman is located 15 miles (25 km) south of Oklahoma City on Interstate I-35.

About 20,000 students are enrolled on the Norman Campus. Almost 85% of them are from Oklahoma. The number of international students is approximately 1,200. In the American school system, "University" is always an institution that consists of different "Colleges". Each student is enrolled in one college and thereby a student of the university. There are 15 colleges on the Norman campus: College of Architecture, College of Arts and Sciences, College of Business Administration, College of Education, College of

Engineering, College of Fine Arts, College of Geosciences, College of Law and College of Liberal Studies. There are also the University College and the Graduate College. Some colleges are further divided into schools and departments. The six medical and health-related colleges are located on the Health Science Center Campus in Oklahoma City.

III- The College of Engineering

The College of Engineering has long promoted Oklahoma's economy through research, instruction and public service. It is the largest engineering program in Oklahoma with 2,300 undergraduate students, 700 graduate students and 100 faculty. It was the first in the USA to offer engineering physics and it is the only public institution in Oklahoma to offer aerospace, petroleum and geological engineering degrees. The College has an excellent engineering computer network (ECN) for research and instruction. Its degrees and continuing education courses are delivered worldwide on-site, by talkback television and satellite systems. All of its programs contribute to the $50 million Sarkeys Energy Center activities. The active faculty has authored 89 books. There are 29 professional society fellows, 51 professionally registered engineers in 14 states, 69 listed in "Engineering Who's Who" and 20 named chairs and professorships.

The College of Engineering is divided into nine schools and departments:

- Aerospace & Mechanical Engineering (AME),
- Civil Engineering & Environmental Science (CEES),
- Chemical Engineering & Materials Science (CEMS),
- Electrical Engineering (EE),
- Computer Science (CS),
- Industrial Engineering (IE),
- Petroleum & Geological Engineering (PGE),
- General Engineering (GE),
- Engineering Physics (EP).

IV- The School of Computer Science

The School of Computer Science became an independent unit in the College of Engineering in 1992. The school has created a broad-based program committed to excellence in teaching, quality research on the leading edge of technology and the professional development of students. Degrees offered by the School of Computer Science are: Bachelor of Science in Computer Science, Master of Science, Doctor of Philosophy and Doctor of Engineering.

Faculty research spans a broad spectrum including:

• Artificial Intelligence-Expert Systems, Knowledge Based Systems, Logic Programming, Computer Vision;

• Database Management-Knowledge Databases and Query Languages, Database Design;

• Operating Systems and Computer Architecture-Scheduling Algorithms, Performance Evaluation, Design Automation for VLSI Architecture;

• Parallel Processing-Large Scale Scientific Computing, Algorithms, Interconnection Networks and Distributed Computing;

• Software Engineering-Software Metrics, Program Verification, Ada Environment and Software Tools,

• Theory of Computing-Computational Complexity, Automata and Formal Languages, Algorithms and Data Structure Theory, VLSI Theory.

Students may participate in many professional activities and organizations that include the student chapter of the Association of Computing Machinery (ACM). Each year students are selected to participate in regional and national professional programming contests.

The School operates a number of special purpose and research laboratories for senior and graduate students. These laboratories are designed to afford experience in particular fields of specialization selected by students.

• Computer facilities

The central computing facility at the University of Oklahoma has an IBM system 3081, which can be accessed from a variety of buildings through remote terminals. The College of Engineering commissioned the Engineering Computer Network (ECN). The nucleus of this system is an Encore minicomputer, supplemented by an array of tape and disk drives. ECN is essentially a terminal-oriented system supporting a variety of programming languages such as C, FORTRAN, Basic, APL, LISP, SNOBOL-4. This system is easily accessible to all Computer Science students. Computer Science and Engineering students can use the system for their regular classwork, special software oriented term projects or thesis work.

• Artificial Intelligence Laboratory

The primary goal of the Artificial Intelligence (AI) Laboratory is to establish a forum in which interdisciplinary research and rapid software prototype development can be achieved through collaborations between faculty members and industrial partners. Facilities at the AI Laboratory include several TI Explorers, PCs and workstations. Present activities are strongly funded by many grants and contracts.

• Parallel Processing Institute

The aim of the Parallel Processing Institute (PPI) is to provide a forum for organized research, education and training on all aspects relating to Parallel Processing. It consists of members of the Computer Science faculty with research interests related to parallel algorithm development, parallel architectures and interconnection networks for parallel computers, VLSI algorithms, etc. Members of the PPI and their students have access to a wide variety of parallel architectures including the Multimax and Alliant at the University of Oklahoma, 128-node Inter Hypercube at the Argonne National Laboratory and the 64-node N CUBE system at the AMOCO Research Center in Tulsa, Oklahoma.

Clustering Algorithms

I- Introduction

1/ OODB concepts [TSANGARIS92b]

The notion of object abstraction was first introduced by object-oriented programming languages. Recently, OODBs have added database functionality to this abstraction as an attempt to increase the modeling power and the applicability of databases. The object-oriented programming language object abstraction is an extension of the data structure concept with the following basic characteristics:

• *structure:* consisting of components that can be atomic (i.e., flat attributes like integers, reals, or strings), objects (i.e., other objects) or object identifiers (i.e., "pointers" to other objects); unlike data structures, the object state (i.e., values of the structure components) is neither directly changeable nor visible to the user;

• *behavior:* determined by methods, predefined fragments of code that manipulate and export the object state (unlike conventional languages that allow arbitrary code to manipulate data structures);

• *type:* prescribing the "structure" and the "behavior" of an object through the specification of its components and its methods;

• *identity:* naming and locating an object in a manner independent of its state; identity is typically supported identifying objects by an unique number, the object identifier (OID); OIDs are assigned by the system at object creation time and cannot be reused, changed or synthesized.

Conceptually, objects can be viewed as vertices of a directed and possibly cyclic graph: the *Object Graph*. The directed edges of the graph represent the object to object references and they are labelled by the names of their components (cf. figure 1).

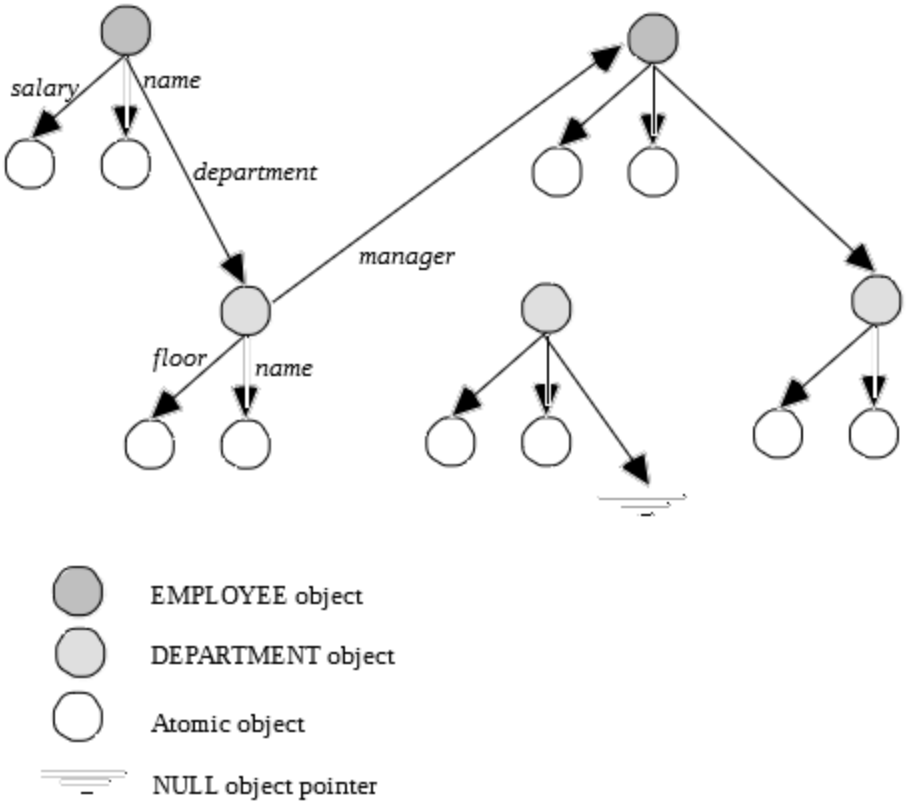

Figure 1: Sample Object Graph

OODBs support additional database functionality:

• *persistence:* i.e., the ability of objects to maintain their state even after the termination of the program that has created them; unlike object-oriented programming languages that only support as many objects as can fit into main memory, OODBs provide access to large collections of objects stored in "stable" secondary storage;

• *storage management:* efficient ways to represent objects in main memory, store them to disk and distribute them to servers; that way OODBs relieve the clients from the burden of storing and retrieving objects from secondary storage, managing main memory as well as the secondary memory;

• *concurrency control and recovery:* to allow concurrent accesses to the objects, and still ensure their integrity; the state of the object base is guaranteed to change only in a consistent manner and is immune to client failures;

• *ad-hoc query facilities:* declarative languages to efficiently apply operations on large sets of objects.

2/ Clustering in OODBs

a) Clustering principles

The goal of object clustering is to reduce the number of disk I/Os for object retrieval. Typically, the unit of data transferred from disk is a page instead of an individual object. If two objects are clustered on the same page, it will take only one disk I/O to access both objects successively. [HURSON93]

Clustering algorithms attempt to improve the performance of object-oriented database systems by placing on the same page related sets of objects [TSANGARIS92a]. In object-oriented databases, complex objects are the basic units of data manipulation. The subobjects of a complex object may come from different classes. Traditional storage systems tend to group records of the same type physically close to each other on disk. This results on tedious and expensive reconstruction procedures (such as join operations) to retrieve complex objects. Therefore, it sounds logical to cluster related objects of different classes together to achieve acceptable performance. [HURSON93]

The problem of clustering can be seen as a graph partitioning problem. The nodes of the graph are the objects and the edges are the links between objects. This problem is NP-complete. However, as the graph of objects represents the database state, all is needed is an incremental solution where new objects are placed at the “right place”. Most of the algorithms used can be classified as greedy algorithms: they scan the objects according to their links and try to place them into the same cluster unit. Thus the cost of clustering has no major impact on the overall system. [BENZAKEN90a]

b) Clustering strategies [CATTELL91a]

Clustering in an OODB can actually be performed in many different ways:

• *composite objects:* objects can be clustered according to aggregation relationships; such clustering may be defined at run time, by identifying particular composite objects to cluster, or the OODB may be given a "hint" in the schema definition that clustering should be performed using particular aggregation relationships;

• *references:* some OODBs allow objects to be clustered according to relationships with other objects; composite objects clustering is, in fact, a special case of this, clustered by the aggregation relationship;

• *object types:* objects may also be clustered by their type; if there is a generalization hierarchy, subtype instances may also be clustered in the same segment; most relational DBMSs cluster by type (relation) by default; however this form of clustering is usually not useful unless repeated access is expected to objects of just one type;

• *indexes:* as in relational DBMSs, it may be possible to cluster objects by an index on their attributes; for example, some documents may be clustered using an index on their title; this can be efficient if documents are accessed frequently in alphabetical order;

• *custom:* some OODBs allow clustering to be performed "on the fly"; in Objectivity/DB, for example, an existing object may be supplied as a parameter to the new-object procedure and the system attempts to create the new object physically close to the existing one.

Unless objects are stored redundantly, an object can generally be clustered according to one of these rules. Where the rules do not conflict, however, it is possible to follow multiple clustering rules. For example, chapters may be clustered within a single segment for objects of type chapter, and within the segment according to the document to which they belong.

This example illustrates that clustering may be performed at two levels:

• *pages:* objects may be clustered according to the smallest physical unit read from disk, which is normally a page; this type of clustering can produce the greatest gains in performance when a "working set" of objects cannot be precisely defined for all applications; page clustering is more useful for clustering by index, reference and composite objects;

• *segments:* objects may be clustered in larger units, when the user is able to specify a meaningful logical grouping for segmentation; segment clustering is most useful for type clustering; it may also be used for composite objects, if used at a sufficiently course grain.

The largest performance gains are generally afforded by page clustering, since pages are the unit of access from the disk and a "working set" of pages is selected dynamically according to the access characteristics of an application program. Segment clustering produces efficiency gains only if relatively large contiguous units are transferred from disk, or when efficiency gains can be made through grouping operations (for example, for composite objects deletion).

c) Users' hints

To expedite the retrieval of related data, database systems often take hints from the user (or database administrator) to store related data physically close together [KIM90b]. For example, the GemStone database administrator, or a savvy application programmer can hint GemStone that certain objects are often used together and so should be clustered on the disk [MAIER86]. The VBASE system allows explicit clustering hints when objects are created [ANDREWS91a]. The strategy adopted in ONTOS is to allow the programmer to specify clustering and to provide tools for reclustering when more experience with the applications permits better choices to be made [ANDREWS91b].

d) Static versus dynamic clustering

In the *static* case, clustering is done at the time objects are created and no reorganization is implied when the links between objects are updated [BENZAKEN90a]. A static clustering scheme offers a good placement policy for complex objects but does not take into account the dynamic evolution of objects. In applications such as design databases, objects are constantly updated during early parts of the design cycle. Frequent updates may destroy the initially clustered structure. To keep the object structure optimized, reorganization might be necessary for efficient future accesses [DEUX90].

Dynamic clustering is done at run time when objects are accessed concurrently and becomes attractive in an environment where the read operations dominate the write operations [BENZAKEN90a]. A dynamic clustering scheme should try to recluster scattered access cost becomes too high. However, reclustering will generate overhead such as extra disk I/O, so it is important to determine when a reorganization should occur. If the overhead is not justified, reclustering may actually degrade the performance [CHENG91].

II- Cactis Clustering Algorithm

1/ Algorithm presentation [HUDSON89]

Cactis is an object-oriented, multi-user DBMS developed at the University of Colorado. It is designed to support applications that require rich data modeling capabilities and the ability to specify functionally-defined data.

The Cactis clustering algorithm is designed to place objects that are frequently referenced together into the same block (i.e., page, i.e., I/O unit) on secondary storage. It can improve response time up to 60%.

In order to improve the locality of data references, data is clustered on the basis of usage patterns. A count of the total number of times each object in the database is accessed is kept, as well as the number of times each relationship between objects in the process of attribute evaluation or marking out-of-date is crossed. Then, the database is periodically reorganized on the basis of this information. The database is packed into blocks using the greedy algorithm shown in figure 2.

This clustering algorithm is also implemented in the Zeitgeist system [FORD88].

Repeat
    Choose the most referenced object in the database that has not yet been assigned a block.
    Place this object into a new block.
    Repeat
        Choose the relationship belonging to some object assigned to the block such that:
            (1) the relationship is connected to an unassigned object outside the block and,
            (2) the total usage count for the relationship is the highest.
        Assign the object attached to this relationship to the block.
    Until the block is full.
Until all objects are assigned blocks.

Figure 2: Cactis clustering algorithm

2/ Clustering example

Let us say we want to cluster six objects into blocks of size 10. The objects' characteristics are given by table 1. It gives for each object its size, the number of times it has been accessed, a list of objects with which it is related and the number of times each of these relationships has been crossed.

Object name	Size	Number of times accessed	Relationships	Number of times crossed
O1	7	90	O3 O4	30 80
O2	2	200	O3 O6	70 200
O3	5	80	O1 O2	30 70
O4	6	50	O1 O5	80 100
O5	4	300	O4 O6	100 50
O6	3	170	O2 O5	200 50

Table 1: Objects' characteristics for clustering example with the Cactis algorithm

Algorithm trace:

NEW BLOC O5 selected
O5-O4 relationship selected, O4 selected, block full

NEW BLOC O2 selected
O2-O6 relationship selected, O6 selected
O2-O3 relationship selected, O3 selected, block full

NEW BLOC O1 selected, all objects clustered

3/ Remarks

• The Cactis clustering algorithm is a static algorithm since it is periodically used to recluster the database when the database is idle. This implies that the database is not clustered on first run because no information about the database is available [CHABRIDON92].

• This algorithm does not require users' hints. This is an advantage since no arbitrary choice has to be made by the user [CHABRIDON93]. But it also implies some time overhead (time to compute total number of times each object is accessed and number of times each relationship is crossed) and space overhead (the main memory space used to store the counters grows with the database size). It also raises the problem of getting pertinent statistics about the database.

III- ORION Clustering Method

ORION is a series of next-generation database systems that have been prototyped at MCC (Microelectronics Computer Technology Corp.) as vehicles for research into the next-generation database architecture and into the integration of programming languages and databases [KIM90a]. ORION has been designed for Artificial Intelligence (AI), Computer-Aided Design and Manufacturing (CAD/CAM) and Office Information System (IOS) applications [BANERJEE87].

ORION supports only a simple clustering scheme. Instances of the same class are clustered in the same physical segment (i.e., a number of blocks or pages). Each class is associated with one single segment. [KIM90a]

But ORION also provides direct support for *composite objects*, i.e., objects with a hierarchy of exclusive component objects (cf. figure 3). The hierarchy of classes to which the objects belong is a *composite object hierarchy*. The object-oriented data model, in its conventional form, is sufficient to represent a collection of related objects. However, it does not capture the IS-PART-OF relationship between objects; one object simply references, but does not own, other objects. A composite object hierarchy captures the IS-PART-OF relationship between a parent class and its component classes, whereas a class

hierarchy represents the IS-A relationship between a superclass and its subclasses. [BANERJEE87]

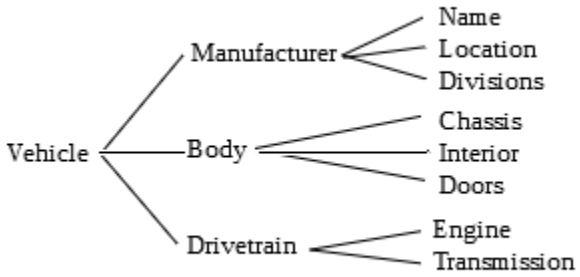

Figure 3: Example of composite object

Then it becomes advantageous to store instances of multiple classes in the same segment. User assistance is required to determine which classes should share the segment. The user can dynamically issue a Cluster message containing a "ListOfClassNames" argument specifying the classes that are to be placed in the same segment. [BANERJEE87]

In ORION, segments have a fixed size. So the number of pages they contain gives the number of I/O necessary to load the segment. When a segment is full, a new page is allocated and linked to the segment (A pointer must be maintained in the segment descriptor.). This implies some overhead to find the address of each additional page [CHABRIDON92].

The advantage of this method is its simplicity that makes the method fast and easy to implement since no cost model is defined and no overhead is implied to determine what is the optimal storage unit for an object. But simplicity also turns to a limitation since users' hints can only be based on the static information given by the data model and not on some information determined by the database usage and which could lead to a better clustering. [CHABRIDON93]

IV- CK Clustering Algorithm

The CK algorithm (from its authors' names: Chang and Katz) is defined in the CAD/CAM context. It can improve response time up to 200% when the Read/Write ratio is high (which is true for real CAD applications) [CHANG89b]. The CK algorithm makes use of several new concepts, such as structural relationships and instance-to-instance inheritance.

1/ Structural relationships

a) Versions

Objects sharing the same interface but having different implementation are called versions [BATORY85]. They represent different design alternatives. E.g., if an object is identified by the pattern: *Name[Version].Type* where "Name" is the object name, "Version" its version number and "Type" its type; Nice[1].car, Nice[2].car and Nice[3].car would be three versions of the same object "Nice" which type is "car".

b) Configurations

A very important characteristic of OODBs is the presence of composite (complex or nested) objects. This concept is represented through composite/component relationships among objects. Coupling the concept of versions with composite objects leads to configurations. A configuration is a composite unit whose components are bound to specific versions (cf. figure 4) [CHANG90].

c) Equivalence

If two objects are alternative representations of the same real world entity, they are equivalent.

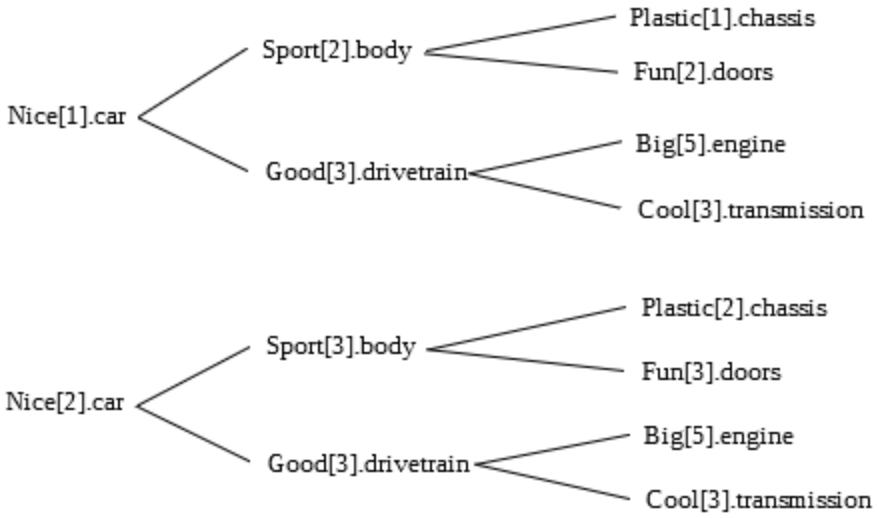

Figure 4: Example of configurations

2/ Instance-to-instance inheritance

Besides structural relationships, inheritance provides additional semantics. As in object-oriented programming languages, a class/subclass hierarchy can be defined for an OODB based on the IS-A relationship. A subclass inherits the structure (i.e., attributes' definitions) and the methods of its superclass. However, in OODBs, this form of inheritance (called type inheritance) is not sufficient. [CHABRIDON92]

The CK algorithm also uses instance-to-instance inheritance that not only transfers the existence of attributes from one object to another (like type inheritance), but moreover the values of these attributes [WILKES88].

Instance-to-instance inheritance is important in computer-aided design databases, since a new version tends to resemble its immediate ancestor. It is useful if a new version can inherit its attributes' values, and more importantly its constraints, from its ancestor. [KATZ91]

3/ **Algorithm Presentation** [CHANG90]

Instance-to-instance inheritance introduces more complexity because it allows attributes to be selectively inherited at run-time. This run-time flexibility requires a sophisticated approach for clustering. The CK algorithm is based on inter-objects access frequencies (given by the user at data type creation time) for each kind of structural relationship, e.g., 20% of access along version relationships, 75% of access along configuration relationships and 5% of access along equivalence relationships.

When a new object is created, the algorithm chooses an initial placement based on which relationship is most frequently used to reach the object (In the above example, a new instance would probably be placed in the same page as its composite objects.). Then, for each inherited attribute, cost formulas are used to choose between implementation by copy or by reference. The augmented access frequencies (i.e., relationship traversal frequencies plus inheritance traversal frequencies) may change the initial placement. The clustering algorithm pseudo code is given by figure 5.

Then, if the best candidate page is full, either the next best candidate page is chosen or the page is split if the expected access cost resulting from the split is an improvement over placement in the next best candidate page.

Page splitting is performed by a greedy algorithm that partitions the inheritance-dependency graph into two sub-graphs that each fit into one page. This algorithm is not optimal, but it is linear (whereas an exact partitioning algorithm would be NP-complete). It is described in figure 6.

4/ **Clustering example**

Let us consider the object hierarchy given by figure 7. Objects are represented according to the following format: *Name[Version].Type* where "Name" is the object name, "Version" its version number and "Type" its type. Numbers above arcs represent the run-time look-up cost of the structural relationship.

We want to cluster these objects in pages of size 10. Table 2 gives types' characteristics. Objects are clustered in their creation order.

```
PROCEDURE cluster_object(target_objet)
BEGIN
    /* step 1: get initial information */
    cluster_policy:=get_policy( );              /* Is page splitting enabled? */
    copy_set:=get_by_copy_set( ); /* Inherited attributes implemented by copy. */
    ref_set:=get_by_ref_set( );                 /* Inherited attributes implemented by reference. */
    inh_page_set:=get_all_inh_page( );      /* Source pages for inherited attributes. */
    struct_page_set:=get_all_struct_page( ); /* Source pages for structural objects. */
    page_set:=inh_page_set+struct_page_set;
    /* step 2: calculate ref_set lookup cost for each page */
    FOR p IN page_set                           /* If by-reference attribute r is */
        FOR r IN ref_set            /* not in page p, storing target object */
            IF r NOT_IN p                       /* in page p requires one run-time */
            BEGIN                               /* lookup for attribute r. */
                weight(p):=1/(prob(p,struct_rel));
                Ref_LookUp(p):=Ref_LookUp(p)+weight(p);
            END;
    /* step 3: calculate copy_set lookup and storage cost for each page */
    FOR c IN copy_set                           /* If by-copy attribute c is not in page */
        FOR p IN page_set                       /* p, we could either cache it in page p */
            IF c NOT_IN p                       /* or change its implementation to be */
            BEGIN                               /* by-reference. */
                weight(p):=1/(prob(p,struct_rel));
                Copy_storage(p):=Copy_storage(p)+size_of(c);
                Copy_LookUp(p):=Copy_LookUp(p)+weight(p);
            END;
    /* step 4:      calculate total cost of every page. If by-copy attributes are */
    /*              implemented by reference, the total cost of storing target object */
    /*              in page p is represented by Total(p,1). Otherwise, the cost */
    /*              is represented by Total(p,2). */
    FOR p IN page_set
        Total_cost(p,1):=Ref_LookUp(p)*Lookup_cost+Copy_LookUp(p)*Lookup_cost;
        Total_cost(p,2):=Ref_LookUp(p)*Lookup_cost+Copy_storage(p)*Storage_cost;
    /* step 5: pick up best candidate page and try to insert the object */
    candidate_page:=Minimum(Total_cost);
    IF (cluster_policy EQ no_split)
        WHILE (NOT_FIT(candidate_page))
            candidate_page:=Next_Min(Total_cost);
    IF ((cluster_policy EQ page_split) AND (NOT_FIT(candidate_page))
        Split_page(candidate_page);
END;
```

Figure 5: Pseudo code for the CK clustering algorithm

The Page_split algorithm assumes that the arc costs C_{ei} (i.e., run-time lookup cost) between objects are always maintained and sorted. The node capacity Cap_{Vi} (i.e., the object size) is also maintained. Subset A and B represent the sets of objects assigned to the new pages after splitting. Both subsets are empty ate the beginning. E is the initial set of arcs relating the objects.

- Step (1): Select the maximum value arc from E as e_{target} and set E to be (E - $\{e_{target}\}$). Let v_{head} and v_{tail} be the head and the tail nodes of e_{target}.
- Step (2): Supposed both v_{head} and v_{tail} are new to subsets A and B. Insert v_{head} and v_{tail} in subset A if Cap_{Vhead} plus Cap_{Vtail} is less than the remaining capacity of subset A. Otherwise, insert v_{head} and v_{tail} in subset B if subset B has space for these nodes. If neither subset A or B could accommodate both v_{head} and v_{tail}, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (3): Supposed v_{head} is in subset A and v_{tail} is not in subset A or B. Insert v_{tail} into subset A if feasible. Otherwise, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (4): Supposed both v_{head} and v_{tail} are visited before, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (5): Look back to step (1) until arc set E is empty.

Figure 6: Page_split algorithm

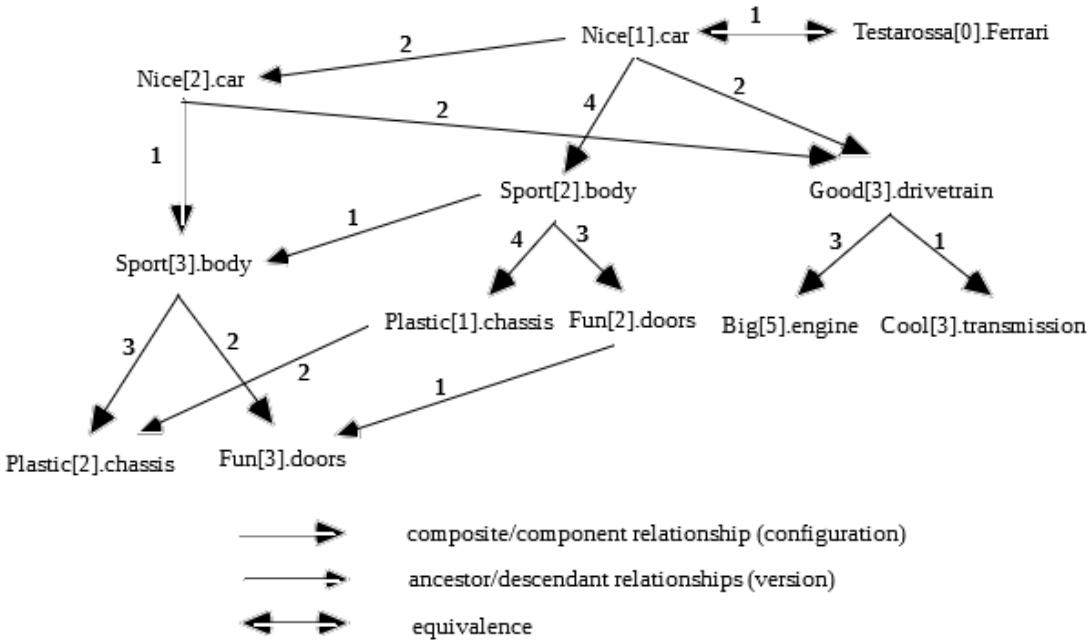

Figure 7: Sample object hierarchy

Type	Object size	Access along versions	Access along configurations	Access along equivalences
Ferrari	3	20 %	10 %	70 %
car	3	65 %	30 %	5 %
body	2	25 %	75 %	0 %
drivetrain	2	30 %	70 %	0 %
chassis	3	40 %	60 %	0 %
doors	1	15 %	85 %	0 %
engine	4	20 %	80 %	0 %
transmission	4	35 %	65 %	0 %

Table 2: Types characteristics for clustering example with the CK algorithm

a) Algorithm trace, Cluster policy = NO PAGE SPLITTING

- Nice[1].car, New page (Page #1), Space left in Page #1 = 7
- Testarossa[0].Ferrari, Page #1, Space left in Page #1 = 4
- Good[3].drivetrain, Page #1, Space left in Page #1 = 2
- Big[5].engine, Page #1 full, no other candidate page => New page (Page #2), Space left in Page #2 = 6
- Cool[3].transmission, Page #1 full, no other candidate page => New page (Page #3), Space left in Page #3 = 6
- Sport[2].body, Page #1, Space left in Page #1 = 0
- Plastic[1].chassis, Page #1 full, no other candidate page => New page (Page #4), Space left in Page #4 = 7
- Fun[2].doors, Page #1 full, no other candidate page => New page (Page #5), Space left in Page #5 = 9
- Nice[2].car, Page #1 full, no other candidate page => New page (Page #6), Space left in Page #6 = 7
- Sport[3].body, Candidate pages = #6 / #1, Page #1 is full => Page #6, Space left in Page #6 = 5
- Plastic[2].chassis, Candidate pages = #6 / #4, Cost formulas give Page #4, Space left in Page #4 = 4
- Fun[3].doors, Candidate pages = #6 / #4, Cost formulas give Page #6, Space left in Page #6 = 4

b) Algorithm trace, Cluster policy = PAGE SPLITTING ENABLED

- Nice[1].car, New page (Page #1), Space left in Page #1 = 7
- Testarossa[0].Ferrari, Page #1, Space left in Page #1 = 4
- Good[3].drivetrain, Page #1, Space left in Page #1 = 2
- Big[5].engine, Page #1 full => PAGE SPLIT (cf. figure 8)
- Cool[3].transmission, Page #1 full => PAGE SPLIT (cf. figure 9)
- Sport[2].body, Page #1 full => PAGE SPLIT (cf. figure 10)
- Plastic[1].chassis, Page #1, Space left in Page #1 = 2
- Fun[2].doors, Page #1, Space left in Page #1 = 1
- Nice[2].car, Page #1 full => PAGE SPLIT (cf. figure 11)
- Sport[3].body, Candidate pages = #5 / #1, Cost formulas give Page #5, Space left in Page #5 = 5
- Plastic[2].chassis, Candidate pages = #5 / #1, Cost formulas give Page #5, Space left in Page #5 = 2
- Fun[3].doors, Candidate pages = #5 / #1, Cost formulas give Page #1, Space left in Page #1 = 0

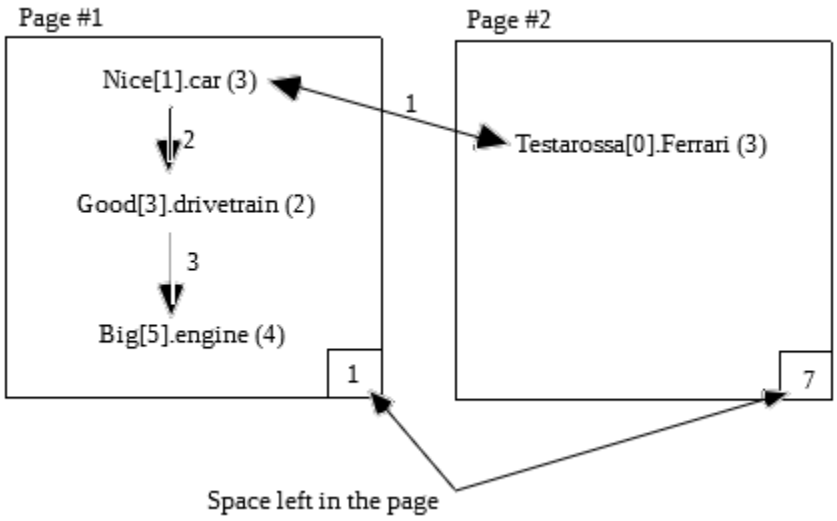

Figure 8: Page splitting #1

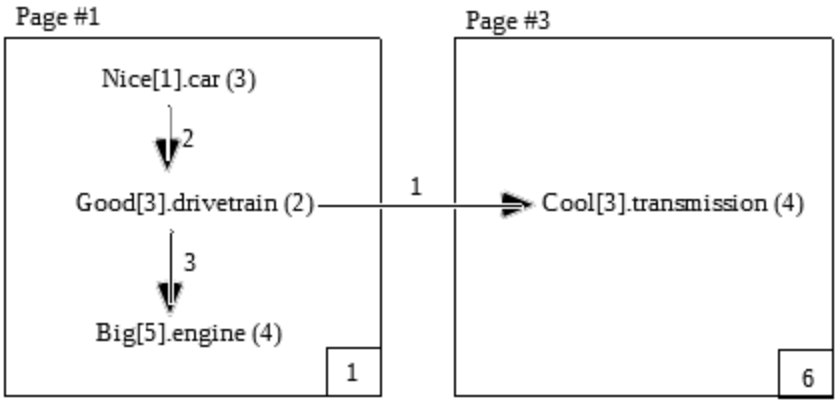

Figure 9: Page splitting #2

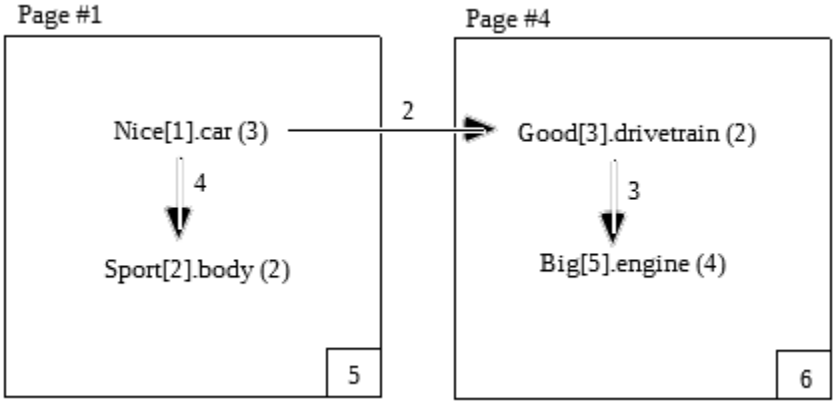

Figure 10: Page splitting #3

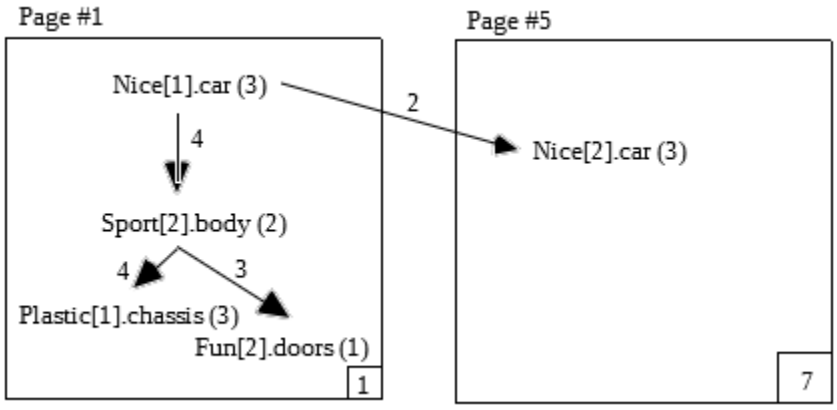

Figure 11: Page splitting #4

OODB Performance Measurement

Performance is a major issue in the acceptance of object-oriented database systems aimed at engineering applications such as Computer-Aided Software Engineering (CASE) and Computer-Aided Design (CAD). There are two main means to evaluate OODBs performance. Either a benchmark can be run on an existing DBMS or an experimental DBMS behavior can be simulated. In both cases, designing the object base and generating queries the major issues. A third way of evaluating in advance the complexity of specific algorithms (such as clustering or buffering algorithms) is mathematical analysis as it is performed in [CHABRIDON92].

I- Performance Benchmarks

1/ The HyperModel Benchmark [ANDERSON90, BERRE91]

The HyperModel Benchmark (also called the Tektronix Benchmark) is based on an extended hypertext model. Hypertext is a generic graph structure consisting of nodes and links. The nodes may contain text or other kinds of data such as bitmaps. The links are used to describe references between the nodes. Hypertext has been proposed as a good model for use in CASE because it is possible to store software and documentation as hypertext graphs.

a) Conceptual schema

Figure 12 provides a conceptual schema for the HyperModel Benchmark using the Object Modeling Technique (OMT).

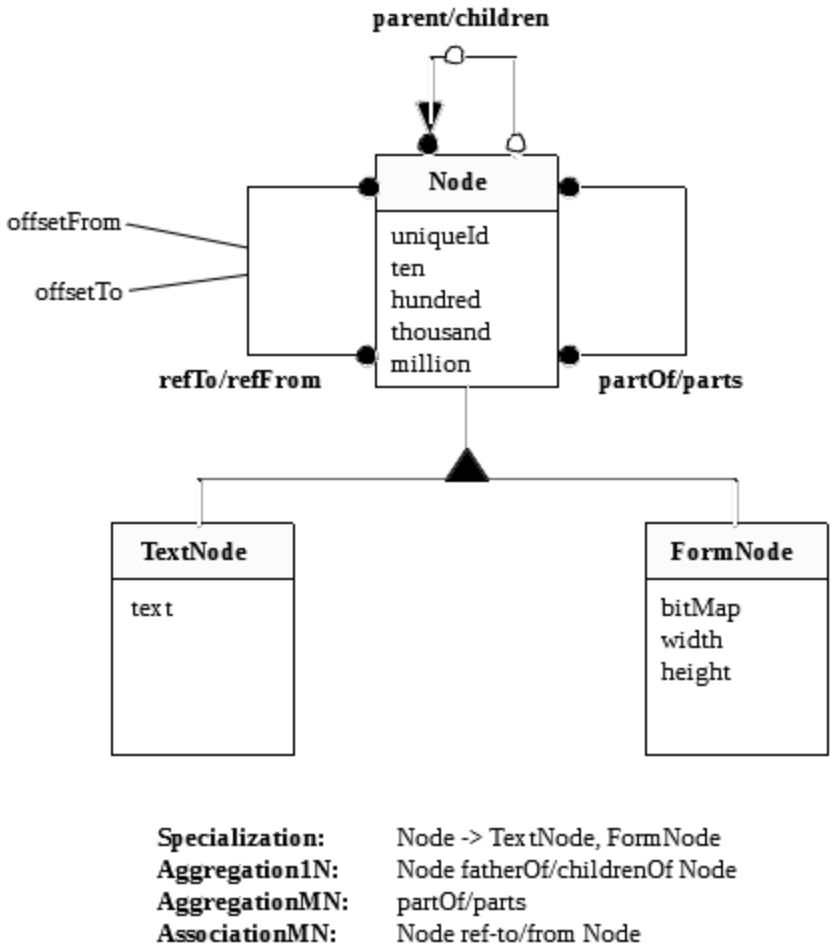

Figure 12: The HyperModel schema

A hypermodel document consists of a number of sections and each section is represented by an object of type *Node*. There are two subtypes of *Node*: *TextNode* and *FormNode*. *Node* is not an "abstract" type; there are *Node* instances which are neither *TextNodes* nor *FormNodes*. There are five attributes (called *uniqueId*, *ten*, *hundred*, *thousand* and *million*) associated with each *Node*. In addition, a *TextNode* has a *text* attribute and a *FormNode* has a *bitmap*, *width* and *height* attribute.

Nodes are interrelated by three relationships: the *parent/children* relationship, the *partOf/parts* relationship and the *refTo/refFrom* relationship. The *parent/children* relationship is 1-N and is used to model the recursive aggregation structure of sections within a document. Furthermore, the children of a given parent are ordered. The *partOf/parts* relationship is an M-N relationship that is constrained to be hierarchical. That is, although parts may share subparts, the *partOf/parts* relationship is acyclic. Finally, the *refTo/refFrom* relationship is an arbitrary M-N relationship.

The *refTo/refFrom* relationship is designed to model Hypertext links, in which attributes are attached to each link to describe the offset (i.e., position) of each endpoint within a node. Associated with each *refTo/refFrom* link are two attributes: *offsetFrom a*nd *offsetTo*. For example, *offsetFrom* might represent the character position within a *TextNode* of the tail of the link.

Considering the *parent/children* relationship, a document can be viewed as a tree. The internal nodes of the tree are instances of the class *Node* while the leaves are either *TextNodes* or *FormNodes* depending on their contents (text or bitmaps).

b) Test database generation

A hypermodel database consists of a single document which is comprised of a network of nodes with the relationships described above. The test database has a fan-out of five nodes at each level in the tree structure described by the *parent/children* relationship (see figure 13). A document will normally contain seven generations.

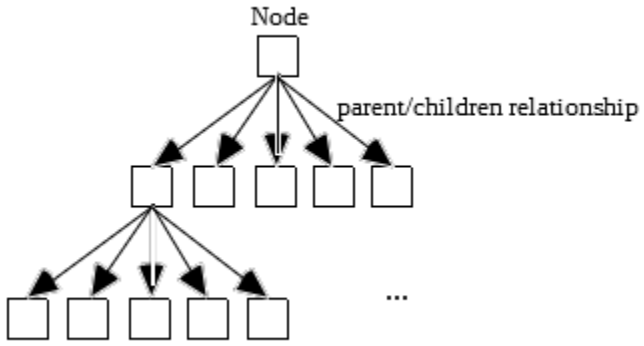

Figure 13: Part of the 7-level parent/children hierarchy

The *partOf/parts* relation is created by relating each node at level *k* to five randomly chosen nodes from level *k+1* in the tree structure created by the *parent/children* relation. The number of *parent/children* relationships and *partOf/parts* relationships are both one less than the total number of nodes. The *refTo/refFrom* relation is created by visiting each node once and creating a reference to another node chosen randomly from the entire tree. The number of *refTo/refFrom* relationships is consequently equal to the number of nodes. The values of *offsetTo* and *offsetFrom* attributes are initialized random integers between 0 and 10.

c) Operations

The benchmark consists of 20 operations. To measure the time to perform each operation, the following sequence must be followed:

Setup: prepare 50 inputs to the operations (the setup is not timed);

Cold Run: run the operation 50 times, on the 50 inputs precomputed in the setup phase; then, if the operation is an update operation, commit the changes once for all 50 operations;

Warm Run: Repeat the operation 50 times with the same inputs to test the effect of caching; again, perform a commit if the operation was an update.

The entire cold run, including the commit (if any), is timed and the mean time per operation is computed by dividing by 50. Likewise, the mean time per operation for the warm run is computed and reported.

The 20 operations are divided into seven kinds:

- Name Lookup Operations: retrieve one randomly selected node;
- Range Lookup Operations: retrieve the nodes satisfying a range predicate based on an attribute value;
- Group Lookup Operations: follow the relationships one level from a randomly selected node;
- Reference Lookup Operations: same as group lookup but in the inverse direction;
- Sequential Scan: all nodes are visited;
- Closure Traversal Operations: similar to Group Lookup Operations but at a predefined depth;
- Editing Operations: retrieve a node and update it.

2/ The CluB-0 Benchmark [BANCILHON92]

CluB-0 was designed to benchmark O_2 clustering algorithm. It is based on the HyperModel Benchmark.

The structure of the CluB-0 Benchmark composition graph is as follows. Only one class *N* is used. The type inheritance graph for this class is shown in figure 14. The database consists of objects of class *N*, called nodes, that form a tree (relationship *T*) with a constant fan-out *f*, overlapping with a graph (relationship *G*). Each internal node of *T*, say a node at depth *i*, is connected to exactly *f* distinct nodes randomly chosen from the set of nodes at depth *i+1* (each branch in the tree has the same depth). Of course, at level 0, the two kinds of links are the same. At any other level, objects can be shared inside the graph induced by *G*.

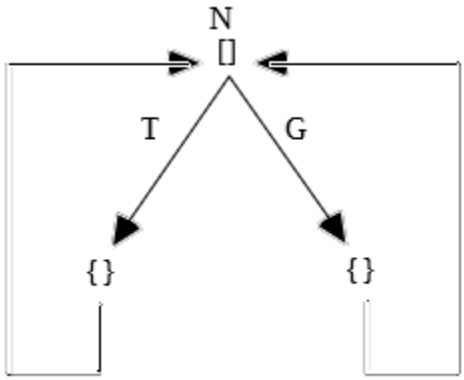

Figure 14: The type inheritance graph for class N

3/ The Synthetic Benchmark [KIM90]

The Synthetic Benchmark is one of the benchmarks that have been used to evaluate ORION performances. A synthetic database is created to perform three sets of ORION operations: basic object manipulation, basic operations on objects with index support, and queries. The database is created for a 10% selectivity for queries.

Basic Object Manipulation

• Create five classes with similar structure (ten attributes) and populate each class with 1000 instances.

• Access all instances of the classes in the order of creation and in random order.

Index Overhead

• Create five classes, each with ten attributes, of which two are indexed; and populate each class with 1000 instances.

• Update all indexed attributes of all instances of the classes in the order of creation and in some random order.

Queries

• Query a class on an indexed attribute.

• Query a class on an non-indexed attribute.

The execution of each operation of this benchmark starts from a cold state (i.e., database restart) and has to be repeated twelve times to reduce the effects of variations.

4/ **The OO1 Benchmark** [CATTELL91b]

This benchmark, named OO1 (Object Operations version 1), sometimes called the "Cattell Benchmark", has been run on a dozen of both relational and object-oriented DBMSs.

a) Database

The OO1 Database is independent of the data model provided by the DBMS. It is defined as two logical records.

```
create table part (  id      integer    not null primary key,
                     type    char(10)   not null,
                     x       integer    not null,
                     y       integer    not null,
                     build   datetime   not null
                     );
```

```
create table connection (
                    from    integer    foreign key reference (part.id),
                    to      integer    foreign key reference (part.id),
                    length  integer    not null,
                    type    char(10)   not null,
                    primary key (from, to, length)
                    );
```

A database of *N* parts will have a dense unique part number (*part.id*) in the range [1..*N*]. Such a database will have 3*N* connections, with exactly three connections from each part to other (randomly selected) parts. The *x*, *y*, and *length* field values are randomly distributed in the range [0..99999], the *type* fields have values randomly selected from the strings {"`part-type0`" ... "`part-type9`"}, and the *build* date is randomly distributed in a ten-year range. The random connections between parts are selected by an algorithm to produce some locality of reference.

b) Measures

The following three operations are the OO1 Benchmark measures. Each measure is run ten times, measuring response time for each run to check consistency and caching behavior.

• *Lookup.* Generate 1000 random part IDs, and fetch the corresponding parts from the database. For each part, call a null procedure written in any host programming language, passing the *x,y* position and type of the part.

• *Traversal.* Find all parts connected to a randomly selected part, or to a part connected with it, and so on, up to seven hops (total of 3280 parts, with possible duplicates). For each part, call a null programming language procedure with the value of the *x* and *y* fields and the part type. Perform the traversal depth-first. Also measure time for *reverse* traversal, swapping "from" and "to" directions, to compare the results obtained.

• *Insert.* Enter 100 parts and three connections from each to other randomly selected parts. Time must be included to update indices or other access structures used in the extension of *lookup* and *traversal*. Call a programming language procedure to obtain the *x,y* position for each insert. Commit the changes to the disk.

II- Simulation

1/ Object Base

In [HE93], two kinds of database generators are proposed. One of them generates totally random graph for the object references. This simulates random user queries for different objects in the system. The object reference requests are distributed randomly over the object cluster. The second request generator creates a Directed Acyclic Graph (DAG). Need for the second request generator arises from the fact that in real world OODB request patterns for the object resemble to a DAG. Once a particular object is referred, probability of a related object to currently referred object is very high and this generates a DAG.

[TSANGARIS92b] used in his simulations the same object base that the one proposed by the CluB-0 Benchmark we described above.

2/ Queries

In simulations performed to evaluate the performances of the CK clustering algorithm [CHANG89b], the checkin and checkout operations are modeled by seven different types of queries:

- simple object lookup,
- component object retrieval,
- composite object retrieval,
- descendant version retrieval,
- ancestor version retrieval,
- corresponding objects retrieval,
- object insertion/deletion/updating.

For example, a checkout operation may consist of several component object retrievals and one corresponding object retrieval. Similarly, a checkin operation invokes several object insertions and updating.

In [TSANGARIS92b], two access models are presented: IID (Independent and Identically Distributed) and SMC/HMC (Simple Markov Chain/Higher order Markov Chain). In IID access model, objects are accessed in a random order. In SMC access model, the probability of accessing an object depends only on the previous objects accessed. Finally, in HMC, the probability of accessing an object depends on the *N* last accessed objects, where *N* is the pre-selected order. Although these access models are used as input by stochastic clustering algorithms, they could also be used to generate access patterns for queries. IID and SMC access models are shown for a set of six objects {a, b, c, d, e, f} by figure 15. Fractions represent the objects' access probabilities.

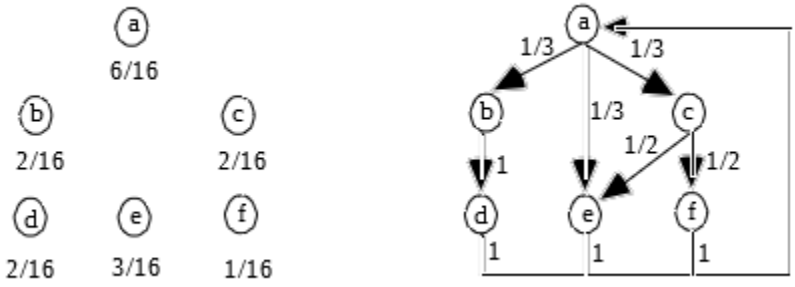

Figure 15: IID and SMC access models

3/ Performance measurements

Response time is measured most of the time [CHANG89b, TSANGARIS92b]. *Number of pages accessed on disk* [TSANGARIS92b], *I/O counts* [HUDSON89] and *average access time per object* [CHENG91] can also be evaluated. In a distributed environment, *network traffic* (i.e., number of objects fetched from other sites) is used [HE93].

[TSANGARIS92a] also introduces a performance metric that expresses the "packing capability" of clustering algorithms: the *Expansion Factor* (EF). Given a set of objects Q that maps to N(Q) distinct pages, EF is defined as:

$$EF(Q) = \frac{N(\Theta)}{\left\lceil \frac{|\Theta|}{\Lambda} \right\rceil}$$

.

where the denominator is the size of ideal packing of ||Q|| objects to pages of L objects each. If each query Q_i is associated with a probability $P(Q_i)$, an average EF can be defined:

$$\overline{EF} = \sum_{i} \Pi(\Theta_i) . E\Phi(\Theta_i)$$

For a given set of queries and their probabilities, the ideal clustering mapping minimizes $\overline{EF}$. Although it is easy to avoid bad clustering mappings of $\overline{EF} = \Lambda$, in general it is very hard to find a clustering mapping of $\overline{EF} = 1$.

Although this definition makes sense for records and queries, EF alone is not an adequate metric for clustering. The EF measures the distribution of objects to pages, but does not take into account the order and frequency with which each object is accessed. There are a large number of possible clustering mappings that have the same EF. However, not all of them achieve the same performance.

III- Clustering Algorithms Comparison Methodology

Several reasons made us choose to use simulation to compare Cactis clustering algorithm, CK clustering algorithm and ORION clustering method. First of all, it is important for the results to be meaningful that the performance evaluation is done using the same "environment" for each algorithm, since we focus specifically on clustering. Therefore we could not have benchmarked each OODB since Cactis and ORION use, for example, different buffering and caching strategies. Furthermore, CK clustering algorithm is not implemented in an OODB yet. Building our own simulation model allows us to ensure that the algorithms are compared in the same conditions.

Mathematical analysis has also already been performed on these three algorithms [CHABRIDON92]. Although it provides exact results, it only gives a general idea of the algorithm performances and cannot detect in which specific cases an algorithm performs better than an other as simulation can.

1/ Object base

Unlike what is done in [TSANGARIS92b], we could not use the kind of database used by benchmarks such as CluB-0. These benchmarks use only a few classes and their schema do not offer as many relationships as we need to model both the class hierarchy (superclass/subclass relationship) and the structural relationships between objects (version, configuration and equivalence relationships). Therefore we decided to use a random object base which class hierarchy forms a DAG, as in [HE93].

The database generation was performed in two phases:
- generate class hierarchies and class definition,
- generate instances for these classes.

a) Class hierarchy generation

Given a number of classes, we first build a class hierarchy that includes versions (1). Then we build a composite hierarchy and add equivalence relationships (2).

(1) A new class is added.
    A random number of versions of this class is added (descendant versions).
    If the new class has a superclass (given a probability of having a superclass) then
        randomly select a superclass among the existing classes,
        inherit attributes and methods of the superclass,
        for each additional version of the class:
            randomly select a superclass among the initial class superclass descendant versions,
            inherit attributes and methods of the superclass.
    Add additional random attributes and methods to all versions
        (sizes of attributes and methods are assigned randomly).
    Compute object size for these classes.

(2) Scan all the classes.
    For each class:
        If it is a component of one class (given a probability of being component) then
            randomly select a class composed of the new class.
        If it has an equivalent class (given a probability of having an equivalent) then
            randomly select an equivalent class.

To simplify the class hierarchy, we did not take into account multiple inheritance because it has no effect on clustering. We also assumed that a given class had one single ancestor version and one single descendant version but could have several component classes or equivalent classes.

b) Instances generation

Instance creation has been designed as a special kind of query. However, the initial database is to be created before any other query can occur, given an initial number of objects. The method we used to generate instances is the following.

For each new object:
Randomly select a class.
If the new object class is a component of another class then
randomly select an instance of this class (if any) to be composed of the new object.
If the new object class is a version then
randomly select one ancestor object in the new object class ancestor class,
If using CK, inherit values of common attributes (either by copy or by reference).
If the new object class has an equivalent class then
randomly select one equivalent object among instances of the equivalent class.

2/ Query generation

a) Transactions

The HyperModel Benchmark [ANDERSON90, BERRE91] provides 20 different types of transactions. From those 20, we have isolated 15 types of transactions (some of them are slightly modified to match the structural relationships we use).

• *Name Lookup*: Retrieve a randomly selected object; fetch one of its (randomly selected) attribute value.

• *Range Lookup*: Select a class at random; select one of its attribute at random; determine randomly two test values; fetch all the attributes of all the instances of the class whose selected attribute value are in the range defined by the test values.

• *Group Lookup*: Given a randomly selected starting object, fetch all the attributes of either:
  - all its component objects,
  - all its equivalent objects,
  - all its descendant versions.

• *Reference Lookup*: Given a randomly selected starting object, fetch all the attributes of either:
  - its composite object,
  - all its ancestor versions.

• *Sequential Scan*: Select a class at random; select one of its attribute at random; fetch this attribute's values for every instance of the class.

• *Closure Traversal*: Given a randomly selected starting object, follow one of the three structural relationships (i.e., version, configuration or equivalence)* to a certain predefined (random) depth D; fetch a random attribute from the resulting object; * the followed relationship can be either always the same or randomly selected.

• *Editing*: Select an object at random; update one of its attribute (randomly chosen) with a random value.

+ *Object Creation*: Creation of a new object (cf. Object base generation). This activates the CK clustering algorithm.

+ *Reclustering*: ORION clustering algorithm needs a "Cluster message" to be dynamically activated [BANERJEE87]. Cactis clustering algorithm is static. We can assume it will also wait for a cluster message before reorganizing the database. However, cluster messages for the Cactis algorithm should be far less frequent than cluster messages for the ORION algorithm since the Cactis clustering algorithm is supposed to run when the database is idle [HUDSON89].

b) Transaction generation

Each kind of transaction should be given a probability of being processed (e.g., 10% for Name Lookup, 7% for Range Lookup, 16% for Group Lookup, etc.). After a random think time, a transaction should be selected according to these probabilities and submitted to the system.

c) Transactions detailed operations

Each transaction is a series of low-level operations (i.e., read or write operations either in memory or on disk). Detailed operations (both I/Os and main memory accesses) necessary to perform the transactions are shown in table 3.

Assumptions:

- Objects OIDs are composed of a class ID and an instance ID, like in ORION [KIM90a].

- Address translation from logical object address (OID) to physical address (physical page address and offset) is performed with three memory accesses to different tables [GRUENWALD91].

- We can access a table showing all pages in the memory buffer. If the buffer's size is B pages, the table will be B memory words in size given that the page number fits in one memory word. The number of memory accesses to scan this table is B at worst, and the average number of memory accesses is B/2.

- Since size of data concerning classes (i.e., all structural links, list of instances, etc.) is small compared to size of instances' data kept on disk, we assume that all data concerning classes is kept in main memory.

- Let A be the attribute average size.

Transaction	Operation	Cost

1/ Name Lookup	Access to object page number Check if page in buffer If page not in buffer Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
2/ Range Lookup	For all N instances of the class: Access to object OID Access to object page number Check if page in buffer If page not in buffer Read attribute value to be tested Compare to test value For all other nA-1 attributes: Read attribute value	N times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses 1 test [nA-1 times: A memory accesses]
3/ Group Lookup along versions	Access to starting object page # Check if page in buffer If page not in buffer For all D descendant objects: Access to descendant OID Access to descendant page # Check if page in buffer If page not in buffer For all nA attributes: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] D times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] nA times: A memory accesses
4/ Group Lookup along configurations	Access to starting object page # Check if page in buffer If page not in buffer For all C component objects: Access to component OID Access to component page # Check if page in buffer If page not in buffer For all nA attributes: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] C times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] nA times: A memory accesses
5/ Group Lookup along equivalences	Access to starting object page # Check if page in buffer If page not in buffer For all E equivalent objects: Access to equivalent OID Access to equivalent page # Check if page in buffer If page not in buffer For all nA attributes: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] E times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] nA times: A memory accesses
6/ Reference Lookup along versions	Access to starting object page # Check if page in buffer If page not in buffer For all N ancestor objects: Access to ancestor OID Access to ancestor page number Check if page in buffer If page not in buffer For all nA attributes: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] N times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] nA times: A memory accesses

Transaction	Operation	Cost
7/ Reference Lookup along configurations	Access to starting object page # Check if page in buffer If page not in buffer Access to composite object OID Access to composite page number Check if page in buffer If page not in buffer For all nA attributes: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] nA times: A memory accesses
8/ Sequential Scan	For all N instances of the class: Access to object OID Access to object page number Check if page in buffer If page not in buffer Read attribute value	N times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
9/ Closure Traversal along versions	Access to starting object page # Check if page in buffer If page not in buffer depth=D, do D times: Access to descendant object OID Access to descendant page # Check if page in buffer If page not in buffer For last object accessed: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] at most D times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
10/ Closure Traversal along configurations	Access to starting object page # Check if page in buffer If page not in buffer depth=D, do D times: Access to *one* component OID Access to component page # Check if page in buffer If page not in buffer For last object accessed: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] at most D times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
11/ Closure Traversal along equivalences	Access to starting object page # Check if page in buffer If page not in buffer depth=D, do D times: Access to *one* equivalent OID Access to equivalent page # Check if page in buffer If page not in buffer For last object accessed: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] at most D times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses

Transaction	Operation	Cost
12/ Random Closure Traversal	Access to starting object page # Check if page in buffer If page not in buffer depth=D, do D times: Access to one descendant, component or equivalent OID Access to this object page # Check if page in buffer If page not in buffer For last object accessed: Read attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] at most D times: 1 memory access 3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
13/ Editing	Access to object page number Check if page in buffer If page not in buffer Write new attribute value	3 memory accesses B/2 memory accesses, B/2 tests [1 I/O] A memory accesses
14/ Object Creation	Cluster new object	Depends on the clustering algorithm used
15/ Reclustering	Cluster database	Depends on the clustering algorithm used

Table 3: Transactions detailed operations

3/ Terminology

Let us state what terms we will use in the following chapters.

- A class *superclass* is its parent class along the class hierarchy.
- A class *subclass* is one of its child class along the class hierarchy.
- A class *ancestor class* is its parent class along the version structural relationship.
- A class *descendant class* is its child class along the version structural relationship.
- A class *composite class* is its parent class along the configuration structural relationship.
- A class *component class* is one of its child class along the configuration structural relationship.
- A class *equivalent class* is one of its target class along the equivalent structural relationship.
- An object *ancestor object* is its previous version.
- An object *descendant object* is its next version.
- An object *composite object* is the object that is composed of it.
- An object *component object* is an object that is a part of it.

  An object *equivalent object* is an object that is an equivalent to it.

SLAM II Simulation Language

SLAM II is an advanced FORTRAN based simulation language that allows models to be built based on three different world views. It provides network symbols for building graphical models that are easily translated into input statements for direct computer processing. It contains subprograms that support both discrete event and continuous model developments. By combining network, discrete event, and continuous modeling capabilities, SLAM II is a **S**imulation **L**anguage for **A**lternative **M**odeling.

I- Introduction to Modeling [PRITSKER86]

1/ Model building

Since a model is a description of a system, it is also an *abstraction* of a system. To develop an abstraction, a model builder must decide on the elements of the system to include in the model. To make such decisions, a purpose for model building should be established. Reference to this purpose should be made when deciding if an element of a system is significant and, hence, should be modeled. A model building approach is presented in figure 16.

2/ The simulation process

The iterative simulation process can be subdivided into the following stages of development:

• *Problem Formulation:* definition of the problem to be studied including a statement of the problem-solving objective;

• *Model Building:* abstraction of the system into mathematical-logical relationships in accordance with the problem formulation;

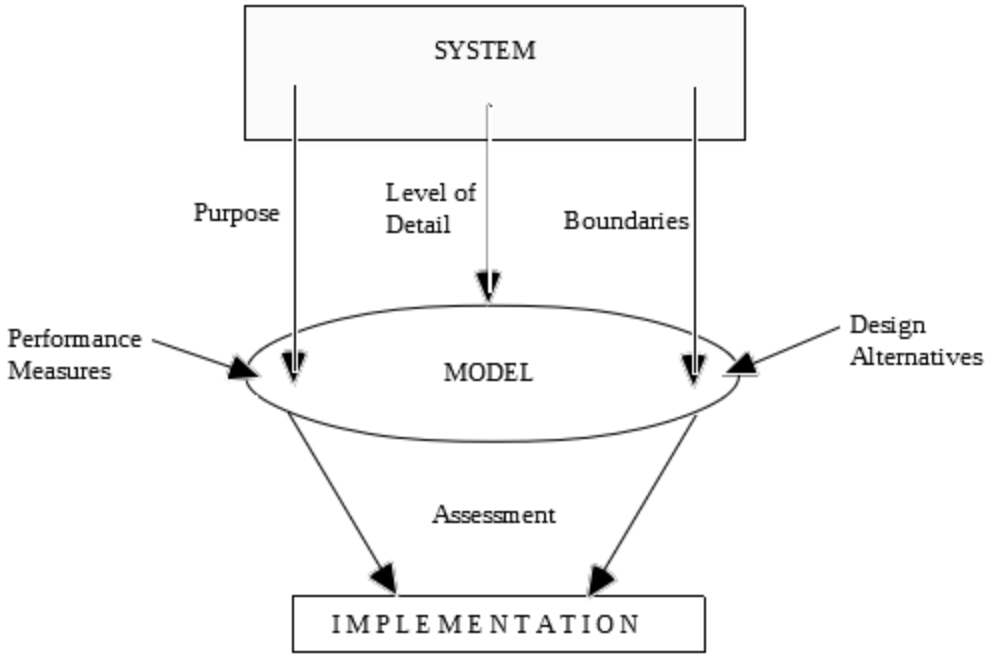

Figure 16: Model building approach

• *Data Acquisition:* identification, specification and collection of data;

• *Model Translation:* preparation of the model for computer processing;

• *Verification:* process of establishing that the computer program executes as intended;

• *Validation:* process of establishing that a desired accuracy or correspondence exists between the simulation model and the real system;

• *Strategic and Tactical Planning:* process of establishing the experimental conditions for using the model;

• *Experimentation:* execution of the simulation model to obtain output values;

• *Analysis of Results:* process of analyzing the simulation outputs to draw inferences and make recommendations for problem resolution;

• *Implementation and Documentation:* process of implementing decisions resulting from the simulation and documenting the model and its use.

The iterative simulation process is summarized by figure 17.

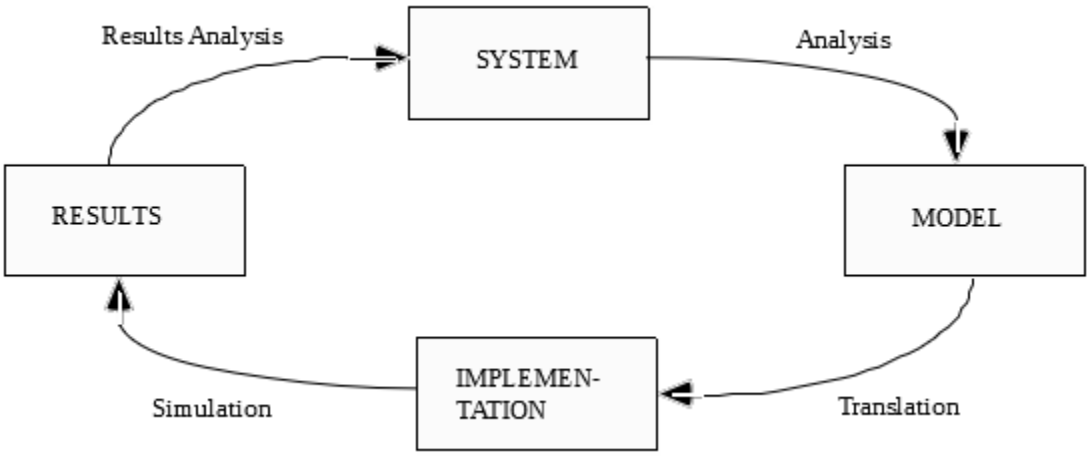

Figure 17: Iterative simulation process

II- Simulation [PRITSKER86]

1/ Simulation Definition

In its broadest sense, simulation is the process of designing a mathematical-logical model of a real system and experimenting with this model on a computer. Thus simulation encompasses a model building process as well as the design and implementation of an appropriate experiment involving that model. These experiments, or simulations, permit inferences to be drawn about systems

- without building them, if they are only proposed systems;
- without disturbing them, if they are operating systems that are costly or unsafe to experiment with;
- without destroying them, if the object of an experiment is to determine their limits of stress.

In this way, simulation models can be used for design, procedural analysis and performance assessment.

2/ The different kinds of simulation

Models of systems can be classified either as discrete change or continuous change. In fact, it may be possible to model the same system with either a discrete change (discrete) or a continuous change (continuous) model. In most simulations, time is the major independent variable. Other variables included in the simulation are functions of time and are the dependent variables. The adjectives discrete and continuous when modifying simulation refer to the behavior of the dependent variables.

a) Discrete Simulation

Discrete simulation occurs when the dependent variables change discretely at specified points in simulated time referred to as event times. The aim of a discrete simulation model is to reproduce the activities that the entities engage in and thereby learn something about the behavior and performance potential of the system. This is done by defining the states of the system and constructing activities that move it from state to state. The state of a system is defined in terms of the numeric values assigned to the attributes of the entities.

A discrete simulation model can be formulated by: 1) defining the changes in state that occur at each event time (*event orientation*); 2) describing the activities in which the entities in the system engage (*activity scanning orientation*); or 3) describing the process through which the entities in the system flow (*process orientation*). The relationship between *event*, *activity* and a *process* is depicted in figure 18.

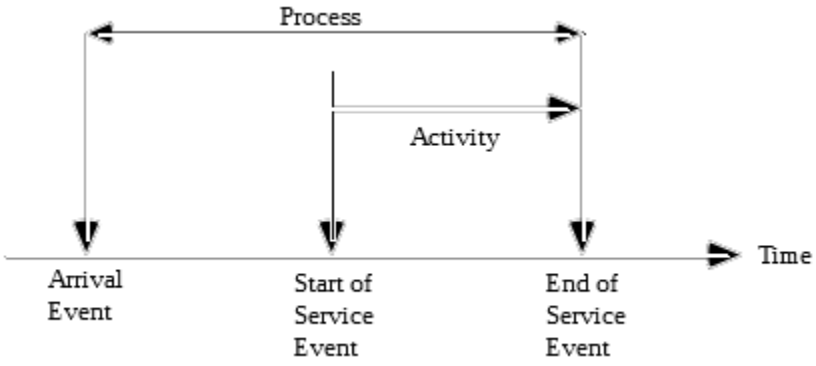

Figure 18: Relationship of events, activities and processes

b) Continuous Simulation

In *continuous simulation*, the dependent variables of the model may change continuously over simulated time. A continuous simulation model is constructed by defining equations for a set of state variables whose dynamic behavior simulates the real system.

Models of continuous systems are frequently written in terms of differential equations of the state variable. The reason for this is that it is often easier to construct a relationship for the rate of change of the state variable than to devise a relationship for the state variable directly. For example, the following differential equation could be used for the state variable s, over time t, together with an initial condition a time 0.

$$\frac{ds(t)}{dt} = \sigma^2(\tau) + \tau^2$$
$$\sigma(0) = \kappa$$

The first equation specifies the rate of change of s as a function of s and t and the second equation specifies the initial condition for the state variable. In some cases, it is possible to determine an analytical expression for the state variable s, given an equation for ds/dt. However, in many cases of practical importance, an analytical solution for s will not be known. As a result, the response s must be obtained by integrating ds/dt over time using an equation of the following type:

$$s(t_2) = \sigma(\tau_1) + \int_{\tau_1}^{\tau_2} \left(\frac{\delta\sigma}{\delta\tau}\right).\delta\tau.$$

Sometimes a continuous system is modeled using difference equations. In these models, the time axis is decomposed into time periods of length Δt. The dynamics of the state variables are described by specifying an equation which calculates the value of the state variable at period k+1 from the value of the state variable at period k. For example, the following difference equation could be employed to describe the dynamics of the state variable s:

$$s_{k+1} = \sigma_k + \rho\,\Delta\tau.$$

When using difference equations, the essential structure of a continuous simulation model is often reflected in the relationship between the state r used to project the state variable at period k+1 from the value s_k of the state variable at period k.

b) Combined Simulation

Combined simulation can also be performed if the dependent variables may change discretely, continuously or continuously with discrete jumps superimposed. There are two types of events that can occur in combined simulations. *Time-events* are those events which are scheduled to occur at specified points in time. They are commonly thought of in terms of discrete simulation models. In contrast, *state-events* are not scheduled, but occur when the system reaches a particular state. The possible occurrence of a state-event must be tested at each time advance in the simulation. For example, as illustrated in figure 19, a state-event could be specified to occur whenever state variable SS(1) crosses state variable SS(2) in the positive direction.

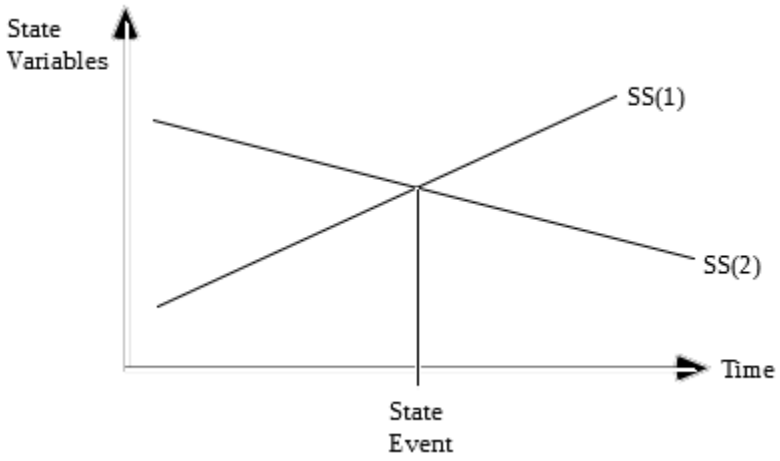

Figure 19: Example of a state-event occurence

In these three cases (discrete, continuous and combined simulation), the time variable may be either continuous or discrete, depending on whether the discrete changes in the independent variable can occur at any point in time or only at specified points.

3/ SLAM II approach

In SLAM II, the alternative modeling world views are combined to provide a unified modeling framework. A discrete change system can be modeled within an event orientation, process orientation or *both*. Continuous change systems can be modeled using either differential or difference equations. Combined discrete-continuous change systems can be modeled by combining the event and/or process orientation with the continuous orientation. In addition, SLAM II incorporates a number of features which correspond to the activity scanning orientation.

The process orientation of SLAM II employs a *network* structure which consists of specialized symbols called *nodes* and *branches*. These symbols model elements in a process such as queues, servers and decision points. The entities in the system flow through the network model.

In the event orientation of SLAM II, the modeler defines the events and the potential changes to the system when an event occurs. The mathematical-logical relationships prescribing the changes associated with each event type are coded by the modeler as FORTRAN subroutines. A set of standard subprograms is provided by SLAM II for use by the modeler to perform common discrete event functions such as event scheduling, file manipulations, statistics collection and random sample generation. The executive control program of SLAM II controls the simulation by advancing time and initiating calls to the appropriate event subroutines at the proper points in simulated time.

A continuous model is coded in SLAM II by specifying the differential or difference equations which describe the dynamic behavior of the state variables. These equations are coded by the modeler in FORTRAN by employing a set of special SLAM II defined storage arrays. When differential equations are included in the continuous model, they are automatically integrated by SLAM II to calculate the values of the state variables within an accuracy prescribed by the modeler.

An important aspect of SLAM II is that alternate world views can be combined within the same simulation model. There are six specific interactions which can take place between the network, discrete event and continuous world views of SLAM II:

- entities in the network model can initiate the occurrence of discrete events;
- events can alter the flow of entities in the network model;
- entities in the network model can cause instantaneous changes to values of the state variables;
- state variables reaching prescribed threshold values can initiate entities in the network model;
- events can cause instantaneous changes to the values of state variables;
- state variables reaching prescribed threshold values can initiate events.

In SLAM II, a sequence of events, activities and decisions is referred to as a *process*. *Entities* flow through a process. This, items are considered as entities. An entity can be assigned *attribute* values that enable to distinguish between individual entities of the same type or between entities of different types. For example, the time an entity enters the system could be an attribute of the entity. Such attributes are attached to the entity as it flows through the network. The resources of the system could be servers, tools or the like for which entities compete while flowing through the system. A resource is busy when processing an entity, otherwise it is idle.

Simulation Model

I- Conceptual model

1/ Overall model

The overall simulation model is inspired by the one provided in [CHANG89a]. It is composed as follows (cf. figure 20).

• Client module: After a predefined think time, the client issues the transactions to the Transaction Manager according to some frequencies of appearance.

• Transaction Manager module: The transaction manager extracts from the transactions which objects have to be accessed or updated, and performs the operations. In the case of a regular operation, object requests are sent to the Buffering Manager. In the case of instance creation or a Cluster message, the Clustering Manager is invoked.

• Buffering Manager: The Buffering Manager checks if an object is in main memory and requests it from disk to the I/O Subsystem if it is not. It also deals with page replacement strategies.

• Clustering Manager: The Clustering Manager is activated depending on the algorithm (i.e., Cactis, CK or ORION) it implements. It deals with reorganizing the database on secondary storage to achieve better performance.

• I/O Subsystem: This module deals with physical accesses to secondary storage.

In the following paragraphs are presented a pseudo-code description for each of the overall model modules.

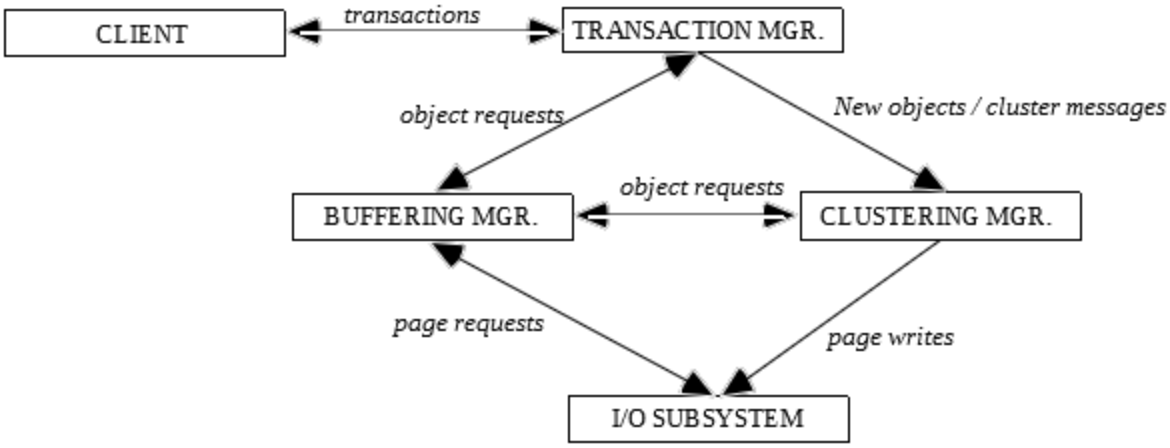

Figure 20: Overall simulation model

2/ Client module (cf. figure 21)

```
think_time:=rnd(avg_think_time);                    /* determine think time at random */
wait(think_time);
transtype:=rnd(set_of_probabilities);               /* determine transaction type at random */
starting_object:=rnd(number_of_objects);            /* determine starting object at random */
submit(trans_type,starting_object);                 /* to the Transaction Manager */
```

Figure 21: Client pseudo code

3/ Transaction Manager module (cf. figure 22)

```
accept(trans_type,starting_object);                 /* from Client */

case trans_type

    1:  begin                                       /* Name Lookup */
            fetch(starting_object);                 /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            attr:=rnd(starting_object.class.number_of_attributes); /* attribute to read selected at random */
            get_value(attr);                        /* read attribute value */
        end

    2:  begin                                       /* Range Lookup */
            attr:=rnd(starting_object.class.number_of_attributes); /* attribute to read selected at random */
            min:=rnd(attr.domain);                  /* minimum range value */
            max:=rnd(attr.domain);                  /* maximum range value */
            for all objects in starting_object.class do
```

```
            fetch(object);                          /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            val:=get_value(attr);                   /* read attribute value */
            if (val≤max and val≥min) then
                for all other attributes do
                    get_value(attribute);           /* read attribute value */
                enddo
            endif
        enddo
    end

3:  begin                                           /* Group Lookup along versions */
        fetch(starting_object);                     /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        while starting_object.descendant≠NIL do
            fetch(starting_object.descendant);      /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            for all starting_object.descendant attributes do
                get_value(attribute);               /* read attribute value */
            enddo
            starting_object:=starting_object.descendant; /* next descendant version */
        enddo
    end

4:  begin                                           /* Group Lookup along configurations */
        fetch(starting_object);                     /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        for all objects in starting_object.list_of_components do
            fetch(object);                          /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            for all object attributes do
                get_value(attribute);               /* read attribute value */
            enddo
        enddo
    end

5:  begin                                           /* Group Lookup along equivalences */
        fetch(starting_object);                     /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        for all objects in starting_object.list_of_equivalents do
            fetch(object);                          /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            for all object attributes do
                get_value(attribute);               /* read attribute value */
            enddo
        enddo
    end

6: begin                                            /* Reference Lookup along versions */
        fetch(starting_object);                     /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        while starting_object.ancestor≠NIL do
            fetch(starting_object.ancestor);        /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            for all starting_object.ancestor attributes do
                get_value(attribute);               /* read attribute value */
            enddo
            starting_object:=starting_object.ancestor;  /* next ancestor version */
```

```
        enddo
    end

7:  begin                                       /* Reference Lookup along configurations */
        fetch(starting_object);                 /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        if starting_object.composite≠NIL then
            fetch(starting_object.composite);   /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            for all starting_object.composite attributes do
                    get_value(attribute);       /* read attribute value */
            enddo
        endif
    end

8:  begin                                       /* Sequential Scan */
        attr:=rnd(starting_object.class.number_of_attributes); /* attribute to read selected at random */
        for all objects in starting_object.class do
            fetch(object);                      /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            get_value(attr);                    /* read attribute value */
        enddo
    end

9:  begin                                       /* Closure Traversal along versions */
        fetch(starting_object);                 /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        depth:=rnd();                           /* depth determined at random */
        d:=depth;
        current_object:=starting_object;
        while (d>0 and current_object.descendant≠NIL) do
            fetch(current_object.descendant);   /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            d:=d-1;
            current_object:=current_object.descendant; /* next descendant version */
        enddo
        if d≠depth then                         /* at least one step performed */
            for all current_object attributes do
                get_value(attribute);           /* read attribute value */
            enddo
        endif
    end

10: begin                                       /* Closure Traversal along configurations */
        fetch(starting_object);                 /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        depth:=rnd();                           /* depth determined at random */
        d:=depth;
        current_object:=starting_object;
        while (d>0 and current_object.list_of_components≠Ø) do
            obj:=rnd(current_object.list_of_components); /* random object in list of components */
            fetch(obj);                         /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            d:=d-1;
            current_object:=obj;                /* next step */
        enddo
        if d≠depth then                         /* at least one step performed performed */
            for all current_object attributes do
```

```
            get_value(attribute);                    /* read attribute value */
        enddo
    endif
end

11: begin                                            /* Closure Traversal along equivalences */
    fetch(starting_object);                  /* request to Buffering Manager */
    wait_for(object_in_memory_message);
    depth:=rnd();                            /* depth determined at random */
    d:=depth;
    current_object:=starting_object;
    while (d>0 and current_object.list_of_equivalents≠Ø) do
        obj:=rnd(current_object.list_of_equivalents); /* random object in list of equivalents */
        fetch(obj);                                  /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        d:=d-1;
        current_object:=obj;                         /* next step */
    enddo
    if d≠depth then                                  /* at least one step performed */
        for all current_object attributes do
            get_value(attribute);                    /* read attribute value */
        enddo
    endif
end

12: begin                                            /* Random Closure Traversal */
    fetch(starting_object);                  /* request to Buffering Manager */
    wait_for(object_in_memory_message);
    depth:=rnd();                            /* depth determined at random */
    d:=depth;
    current_object:=starting_object;
    rel:=rnd(3);                                     /* random structural relationship */
    case rel
        1: test:=(d>0 and current_object.descendant≠NIL); /* follow versions */
        2: test:=(d>0 and current_object.list_of_components≠Ø) /* follow configurations */
        3: test:=(d>0 and current_object.list_of_equivalents≠Ø) /* follow equivalences */
    endcase
    while (test) do
        case rel
            1: obj:=current_object.descendant;
            2: obj:=rnd(current_object.list_of_components);
            3: obj:=rnd(current_object.list_of_equivalents);
        endcase
        fetch(obj);                                  /* request to Buffering Manager */
        wait_for(object_in_memory_message);
        d:=d-1;
        current_object:=obj;                         /* next step */
        rel:=rnd(3);                                 /* next structural relationship */
        case rel
            1: test:=(d>0 and current_object.descendant≠NIL); /* follow versions */
            2: test:=(d>0 and current_object.list_of_components≠Ø) /* follow configurations */
            3: test:=(d>0 and current_object.list_of_equivalents≠Ø) /* follow equivalences */
        endcase
    enddo
    if d≠depth then                                  /* at least one step performed */
        for all current_object attributes do
            get_value(attribute);                    /* read attribute value */
        enddo
```

```
            endif
        end

    13: begin                                          /* Editing */
            fetch(starting_object);                    /* request to Buffering Manager */
            wait_for(object_in_memory_message);
            attr:=rnd(starting_object.class.number_of_attributes); /* attribute to update */
            new_value:=rnd(attr.domain);               /* new random value */
            update_value(attr,new_value);              /* update is done in memory */
        end

    14: begin                                          /* Object Creation */
            new_object.class:=rnd(number_of_classes);  /* randomly select a class */
            if new_object.class.ancestorclass≠NIL then
                ancestor_obj:=rnd(new_object.class.ancestorclass.list_of_instances);
                while ancestor_obj.descendant≠NIL do
                    ancestor_obj:=rnd(new_object.class.ancestorclass.list_of_instances);
                enddo
                ancestor_obj.descendant:=new_object;
                new_object.ancestor:=ancestor_obj;
                for all inherited attributes do
                    if attribute.counter<update_threshold then /* reference attribute */
                        set_value(new_object.attribute,ancestor_obj.OID);
                    else                               /* copy attribute */
                        set_value(new_object.attribute,getvalue(ancestor_obj.attribute));
                    endif
                enddo
            endif
            if new_object.class.compositeclass≠NIL then
                new_object.composite:=rnd(new_object.class.compositeclass.list_of_instances);
                add(new_object,new_object.composite.list_of_components);
            endif
            if new_object.class.equivalentclass≠NIL then
                eq:=rnd(new_object.class.equivalentclass.list_of_instances);
                add(eq,new_object.list_of_equivalents);
                add(new_object,eq.list_of_equivalents);
            endif
        end

    15: begin                                          /* Recluster */
            send(cluster_message);                     /* (to the Clustering Manager) */
        end

endcase
```

Figure 22: Transaction Manager pseudo code

4/ Buffering Manager module

Buffering is not the point we want to evaluate. Hence the buffering strategy we adopted is very simple. Each time a new page has to be brought in memory, we drop the oldest page in buffer and replace it with the new one. If the oldest page had been modified, an I/O is performed to save it on disk. The Buffering Manager pseudo-code is provided by figure 23.

```
accept(object);                                    /* input is an object */
page:=get_page(object);                            /* access to object page number */
if page not in memory then                         /* I/O */
    if oldest_page_in_buffer has been modified then
        write(oldest_page_in_buffer);              /* on disk (request to the I/O Subsystem) */
        wait_for(IO_performed_message);
    endif
    read(page);                                    /* request to the I/O Subsystem */
    wait_for(IO_performed_message);
endif
send(object_in_memory_message);
```

Figure 23: Buffering Manager pseudo code

Note: Since ORION deals with segments rather than with pages, a variant of the Buffering Manager will have to be used in the ORION simulation model. Instead of loading only one page into memory, ORION Buffering Manager will have to load a whole segment.

5/ I/O Subsystem module

A variant of the I/O Subsystem shown by figure 24 will have to be used in the ORION simulation model to take in consideration the fact that a given number of pages in a segment are contiguous. These pages, except the first one, are accessed without seek time.

```
accept(page);                                      /* input is a page */
perform_IO(avg_seek_time+avg_latency_time+transfer_time);
send(IO_performed_message);
```

Figure 24: I/O Subsystem pseudo code

Note: An I/O can be either a read or a write operation.

6/ Clustering Manager module

a) Cactis clustering algorithm implementation

The algorithm shown in figure 25 is a detailed version of the principle algorithm provided in [HUDSON89].

```
wait_for(cluster_message);                              /* when the database is idle */
current:=list();                                        /* current object list (empty) */
for all classes do
    current:=current+class.list_of_instances;           /* current list initialized to all objects */
enddo
repeat
    maxref:=0;
    for all objects in current do
        fetch(object);                                  /* request to the Buffering Mgr. */
        wait_for(object_in_memory_message);
        if object.number_accessed>maxref then           /* new maximum found */
            maxref:=object.number_accessed;
            obj:=object;
        endif
    enddo
    objects_in_page:=list();                            /* list of objects in new page (empty) */
    create(page);                                       /* create new page in memory */
    page.space_left:=page_size;                         /* space_left initialized */
    add(obj,objects_in_page);
    remove(obj,current);
    mcopy(obj,page);                                    /* memory to memory copy */
    repeat
        max_usage:=0;
        for all objects in objects_in_page do
            fetch(object);                              /* request to the Buffering Mgr. */
            wait_for(object_in_memory_message);
            for all relationships in object do          /* i.e., links with ancestor, descendant, */
                                                        /* composite, components and equivalents */
                if (relationship.target_object in current and relationship.usage_count>max_usage) then
                    max_usage:=relationship.usage_count;
                    obj:=relationship.target_object;
                endif
        enddo
        if page.space_left≥obj.class.size then          /* copy object in page */
            fetch(object);                              /* request to the Buffering Mgr. */
            wait_for(object_in_memory_message);
            mcopy(obj,page);
            add(obj,objects_in_page);
            remove(obj,current);
        endif
        page.space_left:=page.space_left-obj.class.size;    /* done all the time to meet the stopping */
                                                            /* condition space_left≤0 */
    until (current=Ø or page.space_left≤0)
until current=Ø
```

Figure 25: Clustering Manager pseudo code - Cactis

Note: A new object is placed in the last page where a new object was last placed if possible or a new page otherwise.

b) ORION clustering method implementation

Since [BANERJEE87, KIM90a] only provide a general clustering method, we had to build an algorithm fitting this method (cf. figure 26).

```
wait_for(cluster_message);                              /* dynamically issued message */
current:=list();                                        /* current object list (empty) */
for all classes do
    current:=current+class.list_of_objects;             /* current list initialized to all objects */
enddo
/* cluster composite objects */
repeat
    stop:=TRUE;                                 /* stop flag */
    obj:=first(current);
    while (stop and obj≠NIL) do
        fetch(obj);                                     /* request to the Buffering Mgr. */
        wait_for(object_in_memory_message);
        if (obj.composite=NIL and obj.component_list≠Ø) then /* composite hierarchy root found */
            stop:=FALSE;
            list:=obj+get_components(obj);              /* get_components is a recursive function */
                                                        /* that returns all obj component objects */
                                                        /*  we get the whole composite hierarchy */
                                                        /* rooted at obj in list */
            current:=current-list;
            create(segment);                            /* create new segment in memory */
            for all objects in list do
                fetch(object);                          /* request to the Buffering Mgr. */
                wait_for(object_in_memory_message);
                mcopy(object,segment);                  /* memory to memory copy */
            enddo
        else
            obj:=next(current);                         /* test on next object */
        endif
    enddo
until stop
/* cluster remaining objects according to their class */
while current≠Ø do
    obj:=first(current);
    create(segment);
    for all objects in current do
        fetch(object);                          /* request to the Buffering Mgr. */
        wait_for(object_in_memory_message);
        if object.class=obj.class then                  /* copy object in segment */
            remove(object,current);
            mcopy(object,segment);
        endif
    enddo
enddo
```

Figure 26: Clustering Manager pseudo code - ORION

The idea of this algorithm is to first cluster objects according to the composite hierarchy they belong to, and then cluster them according to their class. Hence, two types of segments will contain objects either instances of the same class or a whole composite hierarchy.

Note: Since clustering is performed when a Cluster message is sent, a new object is placed in the last segment where a new object was last placed if possible or a new segment otherwise.

c) CK clustering algorithm implementation

The algorithm provided by figure 27 is mostly a reformulating of the [CHANG90] one.

```
accept(object);                                  /* newly created object */
candidate_pages:=list();                         /* empty list */
/* determine the set of candidate pages according to the structural relationships use frequencies */
if (object.class.version%≥object.class.config% and object.class.version%≥object.class.equi%) then
    /* most frequent accesses along versions */
    if (object.descendant≠NIL) then
        add(object.descendant.page,candidate_pages);
    endif
else
    if (object.class.config%≥object.class.version% and object.class.config%≥object.class.equi%) then
        /* most frequent accesses along configurations */
        if (object.composite≠NIL) then
            add(object.composite.page,candidate_pages);
        endif
        for all object component objects do
            add(component.page,candidate_pages);
        enddo
    else                                         /* most frequent accesses along equivalences */
        for all object equivalent objects do
            add(equivalent.page,candidate_pages);
        enddo
    endif
endif
/* select right page using cost formulas if needed */
if (object.ancestor=NIL) then                    /* object is NOT a version -> simple case */
    current:=first(candidate_page);
    while (current.space_left<object.class.size) do   /* while object does not fit into page, skip */
        current:=next(candidate_page);
    enddo
    if (current=NIL) then                        /* no room for the new object */
        if (policy=SPLIT) then
            split(first(candidate_pages),object);      /* the first candidate page is split */
        else
            create(newpage);                     /* Create new page in memory */
            mcopy(object,newpage);               /* object moved in new page */
        endif
    else
        read(current);                           /* direct I/O to get the page */
```

```
        wait_for(IO_performed_message);
        mcopy(object,current);                          /* object moved in page */
    endif
else                                                    /* object IS a version */
    add(object.ancestor,candidate_pages);               /* its ancestor'page becomes candidate page */
    selected_page:=NIL;
    mincost:=∞;
    for all pages in candidate_pages do
        for all attributes of object do
            if attribute is by_reference then
                if attribute.ref_object.page≠page then /* page has a cost */
                    weight:=1/rnd(0,1);                 /* probability of accessing a given page via a */
                                                        /* relationship evolves dynamically, so it is */
                                                        /* determined at random */
                    ref_lookup(page):=ref_lookup(page)+weight;
                endif
            else                                        /* by_copy attribute */
                if object.ancestor.page≠page then               /* page has a cost */
                    copy_storage(page):=copy_storage(page)+attribute.size;
                    weight:=1/rnd(0,1);                 /* probability of accessing a given page via a */
                                                        /* relationship evolves dynamically, so it is */
                                                        /* determined at random */
                    copy_lookup(page):=copy_lookup(page)+weight;
                endif
            endif
        enddo
        /* compute total costs */
        totalcost(page,1):=(ref_lookup(page)+copy_lookup(page))*lookup_cost;
        totalcost(page,2):=ref_lookup(page)*lookup_cost+copy_storage(page)*storage_cost;
        for i:=1 to 2 do
            if totalcost(page,i)<mincost then                   /* new minimum cost */
                selected_page:=page;
                mincost:=totalcost(page,i);
            endif
        enddo
    enddo
    if selected_page=NIL then                                   /* no candidate page found */
        create(newpage);
        mcopy(object,newpage);                                  /* copy object to new page */
    else
        if selected_page.space_left≥object.class.size then      /* enough space to store object */
            fetch(selected_page);                               /* direct I/O */
            wait_for(IO_performed_message);
            mcopy(object,page);                                 /* copy object to page in memory */
        else
            if (policy=SPLIT) then
                split(selected_page),object);                   /* selected page is split */
            else
                create(newpage);                                /* Create new page in memory */
                mcopy(object,newpage);                          /* object moved in new page */
            endif
        endif
    endif
endif
```

Figure 27: Clustering Manager pseudo code - CK

Note 1: The page weight is here assigned at random. This is done because we will have no way in our simulation to know the probability of accessing a page through a given relationship.

Note 2: The CK algorithm needs to know the space left in a given page. We will assume that it is maintained in the page header.

II- Simulation parameters

Parameters are divided into two categories: static parameters that may not change from one simulation to another and dynamic parameters that can vary from one simulation to another.

1/ Static parameters (cf. table 4)

Parameter name	Designation	Value	Justification, References
RAVGTHINK	Average client think time	4 s	[CHANG89a]
RCC	Average locking/unlocking time (concurrency control)	0.5 ms	[SRINIVASAN91]
IMLVL	Multiprogramming level	10	[GRUENWALD91]
IWDSIZE	Size of one memory word	4 bytes	[GRUENWALD91]
ICPU	CPU power	2 Mips	[GRUENWALD91]
RMACC	Memory word access time	0.0001 ms	[GRUENWALD91]
RMTEST	Time for comparing two memory words	0.0007 ms	Two memory accesses, one subtraction
IPGSIZE	Size of disk page	2048 bytes	[CHENG91]
RSEEK	Average disk seek time	28 ms	[CHENG91]
RLATENCY	Average disk latency time	8,33 ms	[CHENG91]
RTRANSFER	Disk page transfer time	1.28 ms	[CHENG91]

Table 4: Static parameters

2/ Dynamic parameters (cf. table 5)

Parameter name	Designation	Default value	Range
NCL	Number of classes	20	10-30
IAVGVER	Average number of versions per class	3	1-5
RPSUPER	Probability for a class of having a superclass	0.9	0-1
RPCOMP	Probability for a class of being a component class	0.5	0-1
RPEQUI	Probability for a class of having an equivalent class	0.1	0-1
INOBJ	Initial number of objects	400	100-1000
IAVGASIZE	Average attribute size	1 word	1-3 words
IAVGNATTR	Average number of attributes per class	10	5-20
IBUFF	Size of memory buffer	10 pages	10-100 pages
IMD	Maximum depth in Closure Traversals	5	3-10
ISEGSIZE	Default segment size (ORION)	5	3-10
ITHRESHOLD	Update Threshold (CK)	25	0-255
ISCALEF	Scale factor (CK)	0.5	0-1
ISPLIT	Page splitting policy (CK)	ON	ON/OFF
PT1	Probability of Name Lookup	0.065	0-1
PT2	Probability of Range Lookup	0.065	0-1
PT3	Probability of Group Lookup along versions	0.065	0-1
PT4	Probability of Group Lookup along configurations	0.065	0-1
PT5	Probability of Group Lookup along equivalences	0.065	0-1
PT6	Probability of Reference Lookup along versions	0.065	0-1
PT7	Probability of Reference Lookup along configurations	0.065	0-1
PT8	Probability of Sequential Scan	0.065	0-1
PT9	Probability of Closure Traversal along versions	0.065	0-1
PT10	Probability of Closure Traversal along configurations	0.065	0-1
PT11	Probability of Closure Traversal along equivalences	0.065	0-1
PT12	Probability of random Closure Traversal	0.065	0-1
Parameter name	**Designation**	**Default value**	**Range**

PT13	Probability of Editing	0.1695 (Cactis) 0.169 (ORION) 0.17 (CK)	0-1
PT14	Probability of Object Creation	0.05	0-1
PT15	Probability of Reclustering	0.0005 (Cactis) 0.001 (ORION) 0 (CK)	0-1

Table 5: Dynamic parameters

Note: Transactions default probabilities are computed on the basis of a Read/Write percentage of about 80% / 20%. All "reading" transactions (i.e., transactions number 1 to 12) are given the same probability to occur.

III- SLAM II Implementation

We used SLAM II version 3.1 and FORTRAN 77 to implement our simulation models. Three simulation models have been written, each of them implementing one of the studied clustering algorithms. In these models, we have separated the object base management from the simulation aspects. Since we needed a complex and dynamic data structure for the object base, the object base data structure and the database operations were coded in FORTRAN. On the other hand, all that deals with simulation is part of the SLAM II model. The SLAM II model is a direct translation of the conceptual model. It also includes the definition of all the simulation parameters. The FORTRAN-SLAM II interface is constituted of a set of USERF functions. The SLAM II-FORTRAN interface is performed through the SLAM II common variables. All listings are provided in [DARMONT94].

Simulation Results

I- Performance measurements

For each case we studied (i.e., each configuration of the dynamic parameters' values), we performed ten simulations in a row without reinitializing the random seed so that we get proper mean results. Simulated time for each single simulation run was three hours.

The comparison criteria we adopted are the following:

• **Response Time:** response time is measured for all transaction types except reclustering (which is considered as a special transaction in the Cactis and ORION simulation models; however, time when transactions are blocked because of a reclustering is also taken into account); it is a good metric for overall performance;

• **Transactions I/Os:** transactions I/Os is the number of I/Os performed to complete regular transactions; transactions I/Os may be an indication on how well objects are clustered;

• **Clustering Time Overhead:** clustering time overhead measures the time needed to reorganize the database; it includes I/O time and the time necessary to perform the memory operations needed by the clustering algorithm but it does not take into account the counters updates performed by Cactis and CK since those take a negligible amount of time compared to even one single I/O;

• **Clustering I/O Overhead:** clustering I/O overhead is the number of I/Os performed during database reorganizations and object clustering;

• **Maximum number of pages used:** maximum number of pages used is the maximum number of disk pages needed by a clustering algorithm to cluster all the objects of the database;

• **System Throughput:** system throughput is the number of transactions completed per second.

II- Results

1/ Effect of the number of objects in the database

We first tested the effect of varying the initial number of objects in the database using a uniform random distribution to choose the transactions' starting objects. This is not always realistic since there may be objects that are "hotter" (i.e., more frequently accessed) than others in real world applications. Furthermore, the performance of the Cactis clustering algorithm depends on run-time computed statistics, such as object's access frequencies, that are not the same when using different random distributions when selecting the transactions starting object. So we implemented in a second series of simulations a normal random distribution for the transactions' starting objects, which is very similar to the "skewed random" function introduced in [TSANGARIS92b].

a) Uniform distribution for starting object

Figure 28a shows that response time when using Cactis is 24% lower than when using ORION. Figure 28b shows that, for CK, response time increases linearly with the number of objects and that CK algorithm totally outperforms the other two (being about 800 times better than Cactis). "PS ON" and "PS OFF" stand for page splitting used or not.

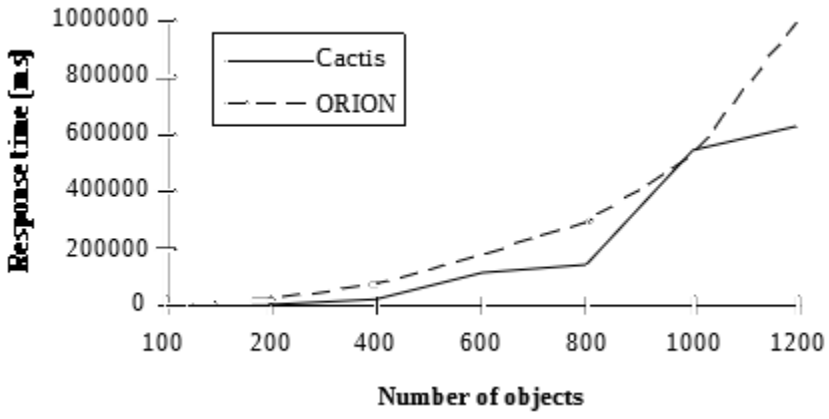

Figure 28a: Response time function of number of objects (U)

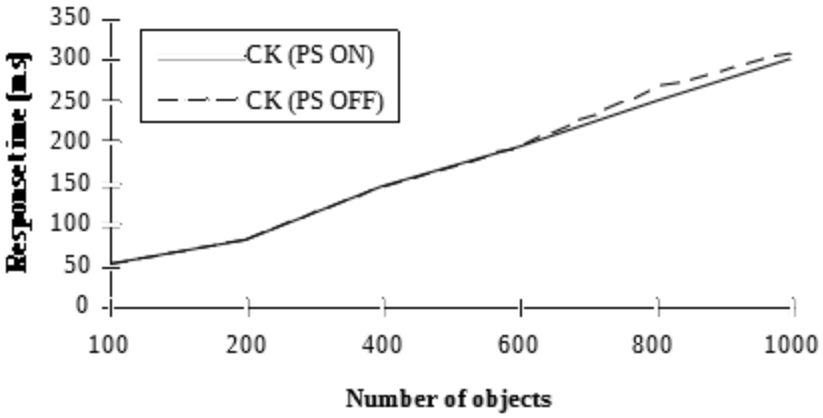

Figure 28b: Response time function of number of objects (U)

Transactions I/Os giving an idea of how good is a clustering scheme, figure 29 shows that objects are 3.8 times better clustered by Cactis and CK (Cactis being slightly better) than they are by ORION. It seems surprising that Cactis clusters so well and performs worse than CK, but figures 30a, 30b and 31a, 31b show again that CK outperforms both Cactis and ORION in terms of low clustering overhead (being 266 times better than Cactis and 502 times better than ORION). Furthermore, clustering overhead is almost constant for CK. Such an outstanding performance is due to the true dynamic nature of CK, which is called only at object creation time and only accesses the object to cluster related objects once. Variations in clustering overhead come from variations in the number of created objects.

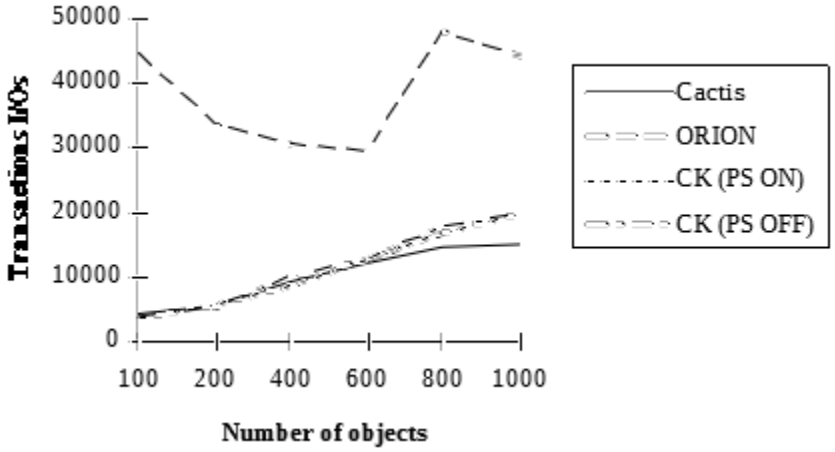

Figure 29: Transactions I/Os function of number of objects (U)

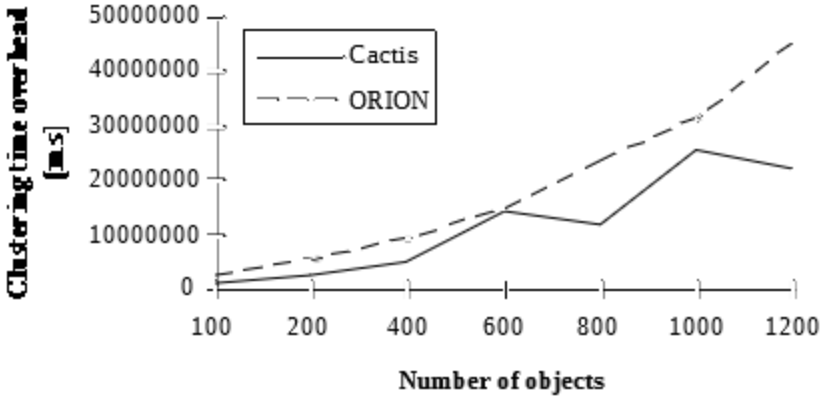

Figure 30a: Clustering time overhead function of number of objects (U)

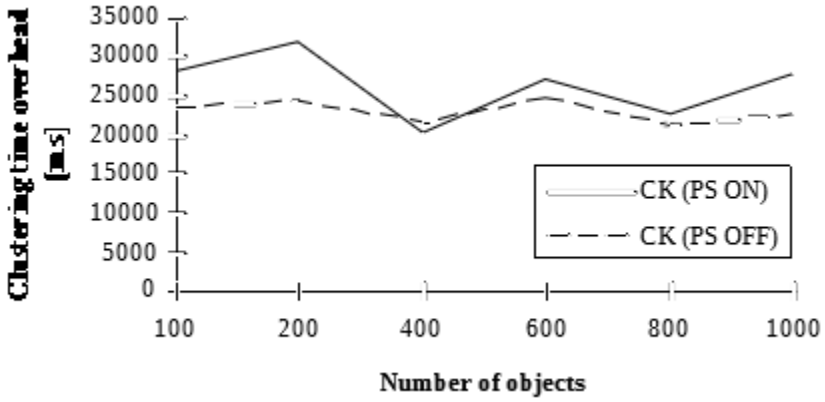

Figure 30b: Clustering time overhead function of number of objects (U)

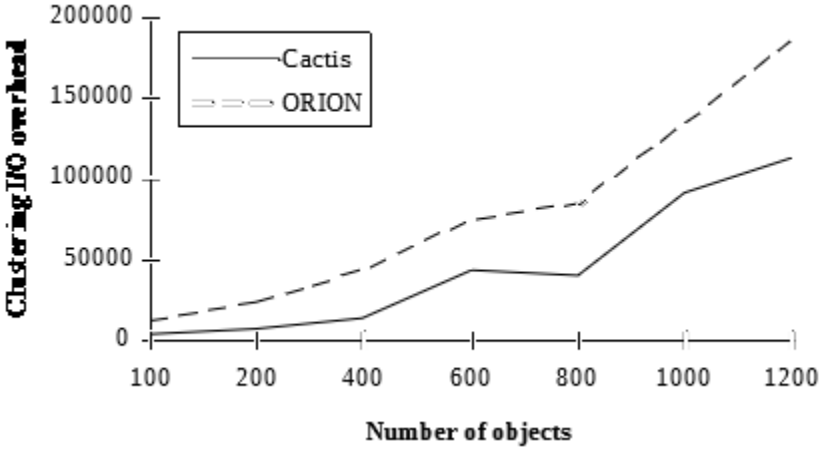

Figure 31a: Clustering I/O overhead function of number of objects (U)

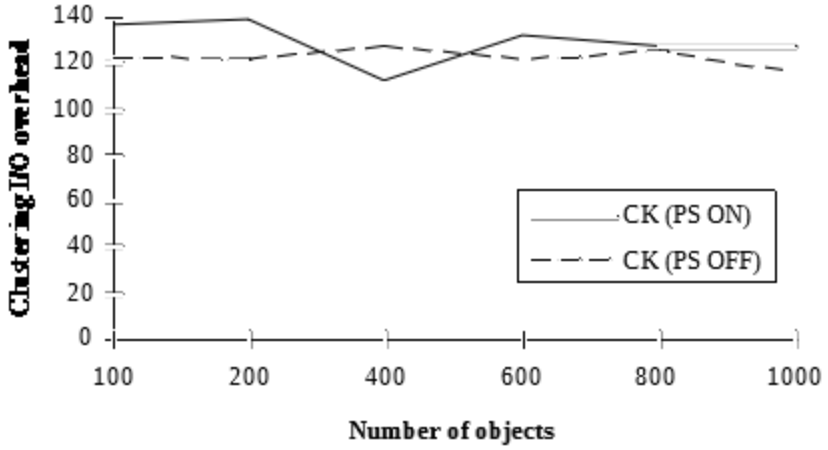

Figure 31b: Clustering I/O overhead function of number of objects (U)

The more a clustering algorithm is "sophisticated" (i.e., the more it clusters object according to precise rules), the more it is likely to use a greater amount of disk pages to cluster the object base. The maximum number of disk pages used (as shown by figure 32), as expected, is higher for "sophisticated" algorithms, i.e., CK needs 1.8 times as many pages as Cactis and Cactis needs 1.3 times as many pages as ORION, for which number of pages increases linearly.

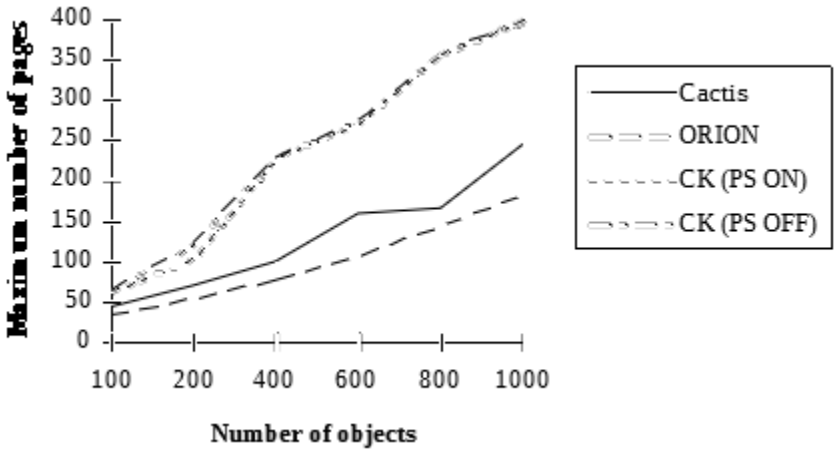

Figure 32: Maximum number of pages used function of number of objects (U)

Since average client think time (i.e., time between two transaction generations) is 4 seconds, optimal throughput lies around 0.25 transactions per second. Figure 33 is coherent with figures 28a and 28b, showing a near constant throughput for CK.

Throughput is high for all algorithms because a typical transaction is executed in much less time than the average think time.

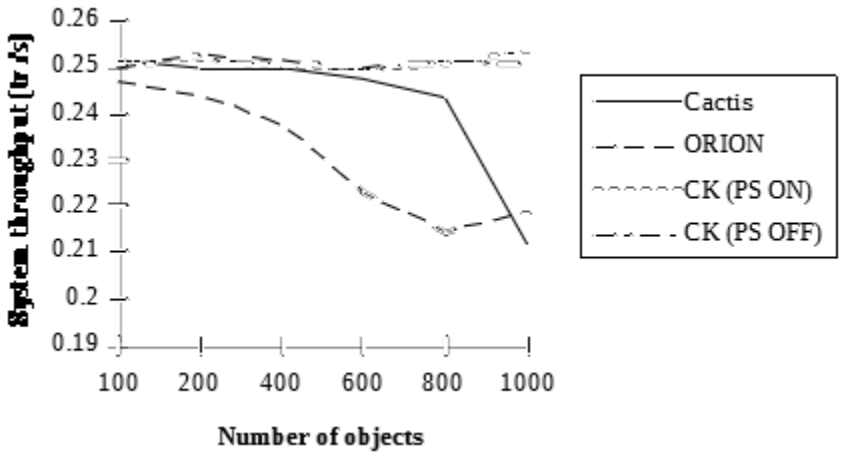

Figure 33: System throughput function of number of objects (U)

Note: Though page splitting should allow a better performance, our simulations show that CK algorithm performances are very close whenever using the page splitting policy or not. This is due to our implementation of the page splitting algorithm that is not as efficient as it could be in reality, because we had no way to know or compute lookup costs in our simulations. Hence we used random lookup costs and thus could not achieve optimal object placement.

b) Normal distribution for starting object

Figures 28a and 34a show indeed that Cactis performs 1.5 times better when objects are not accessed through a uniform distribution, especially for intermediate numbers of objects (between 400 and 600).

On the contrary, as shown by figures 36a, 36b, 37, 38a, 38b, 39a and 39b, CK and ORION algorithms does not show any significative change of performance since they do not use such statistics.

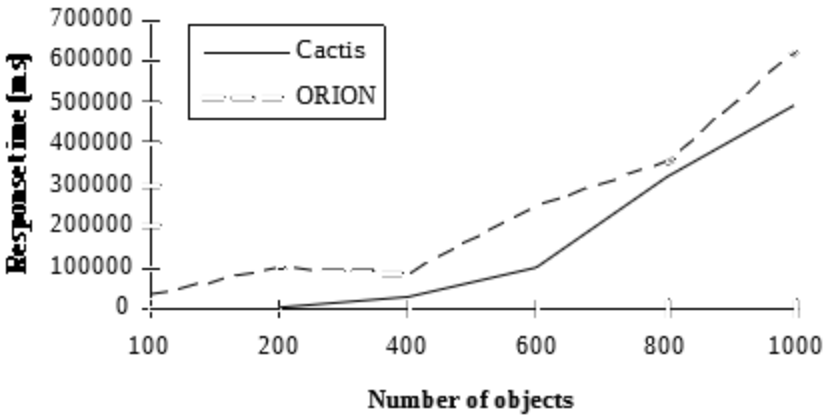

Figure 34a: Response time function of number of objects (N)

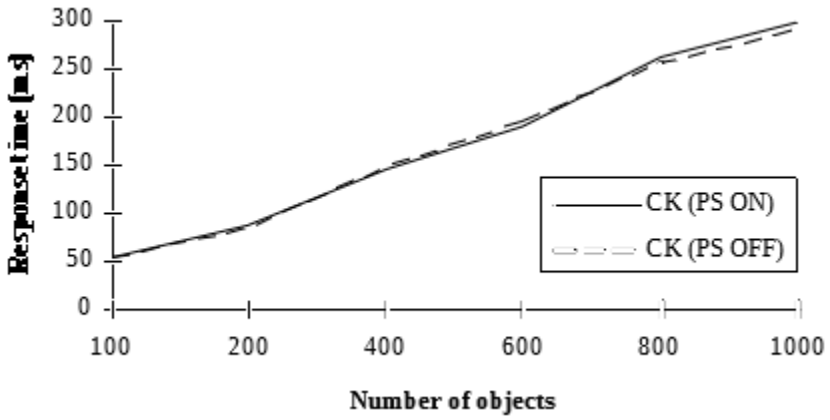

Figure 34b: Response time function of number of objects

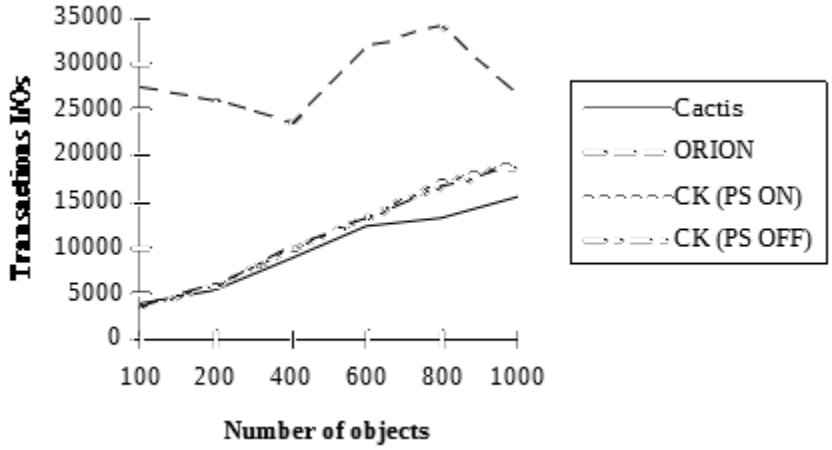

Figure 35: Transactions I/Os function of number of objects (N)

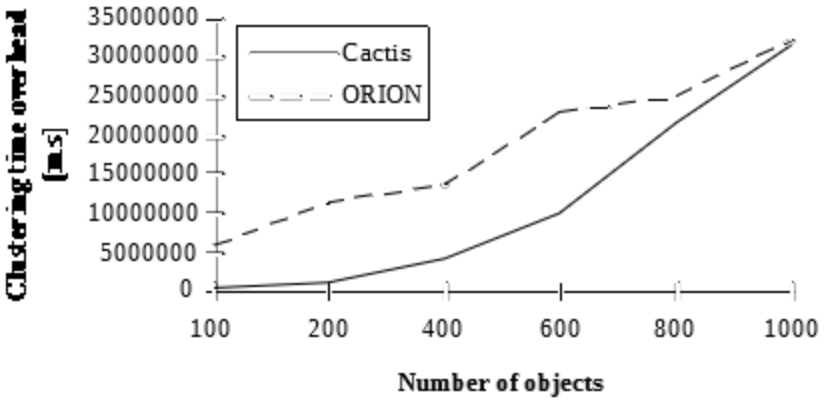

Figure 36a: Clustering time overhead function of number of objects (N)

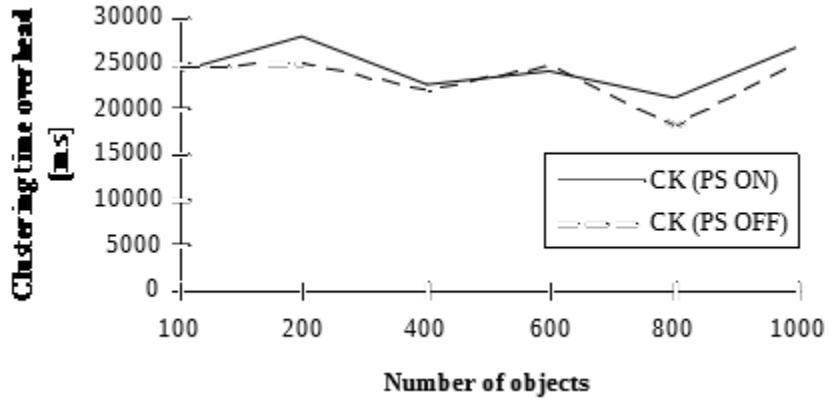

Figure 36b: Clustering time overhead function of number of objects (N)

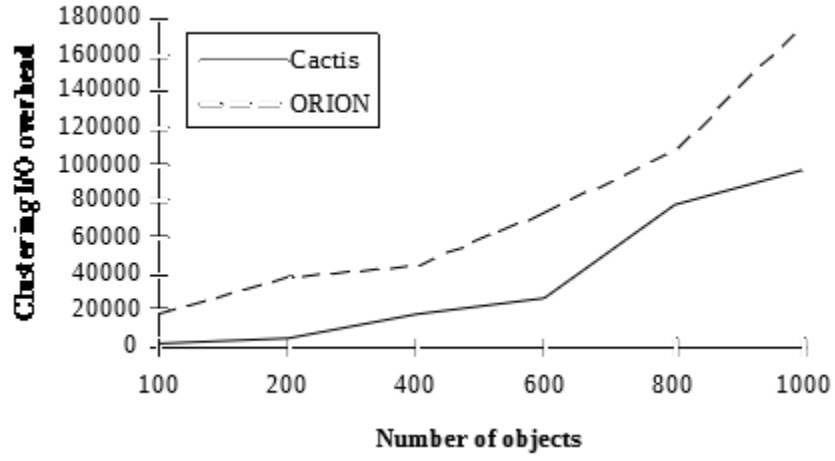

Figure 37a: Clustering I/O overhead function of number of objects (N)

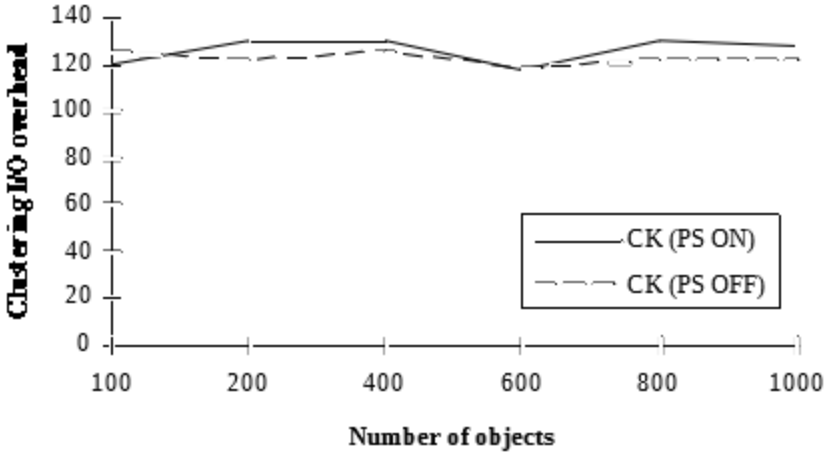

Figure 37b: Clustering I/O overhead function of number of objects (N)

Figure 38 shows the number of pages used by the clustering algorithms. As expected (number of pages is independent of workload), it does not vary when switching from uniform to normal distribution for starting object.

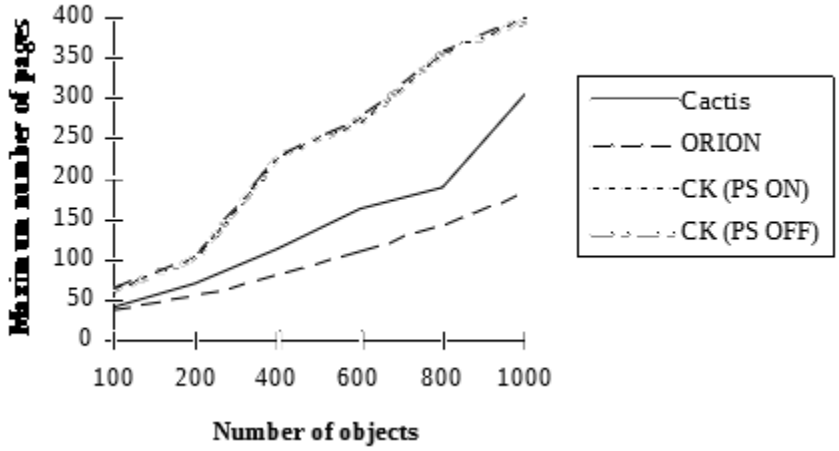

Figure 38: Maximum number of pages used function of number of objects (N)

Figure 34a, 34b and 39 show that system throughput still matches response time.

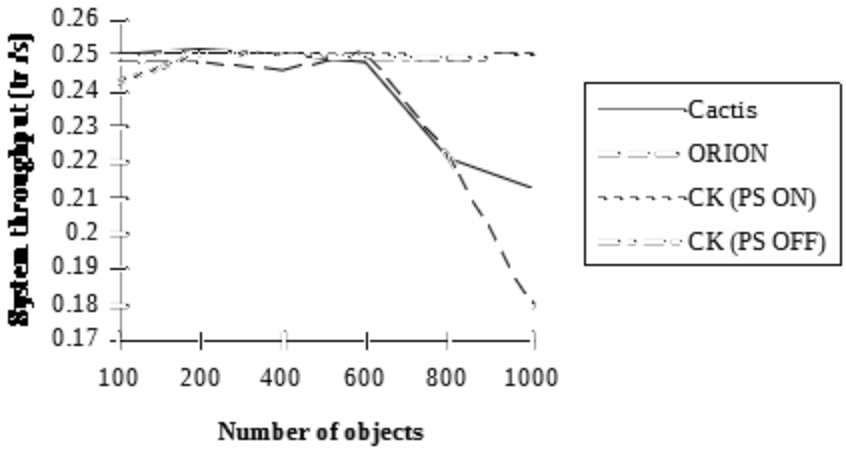

Figure 39: System throughput function of number of objects (N)

2/ Effect of the memory buffer size

On one hand, increasing the buffer capacity may lessen the effects of clustering since a given set of related objects has a higher probability of being in main memory instead of on secondary storage. On the other hand, objects are also accessed when reorganizing the database. Thus, increasing buffer capacity should decrease clustering overhead, especially for the ORION clustering algorithm that may make several non-consecutive accesses to each object. We performed this set of simulations using an initial database of 400 objects and selecting the transactions' starting objects from a uniform random distribution.

Figures 40a and 40b show that response time decrease linearly with the buffer size for Cactis and CK algorithms. The dual effect of increasing the buffer capacity can be seen on figures 41, 42a, 42b, 43a and 43b where both transactions I/Os and clustering overhead decrease linearly (still for Cactis and CK). The ORION algorithm has a similar behavior, but the gain in performance is felt earlier than with the other algorithms and then the gain in performance is less important (cf. figure 40a). We can explain this by the fact that the ORION algorithm uses a smaller amount of pages than the other algorithms to cluster the database. Thus, the buffer size grows faster relatively to the database size. (For instance, a buffer size of 20 pages represents 25% of the database size for ORION versus only 15% for Cactis and 9% for CK.) Figures 40a, 42a and 43a present rather big variations in performance when buffer capacity grows over 40 pages. However, these values oscillate around a mean value that decreases slowly but linearly.

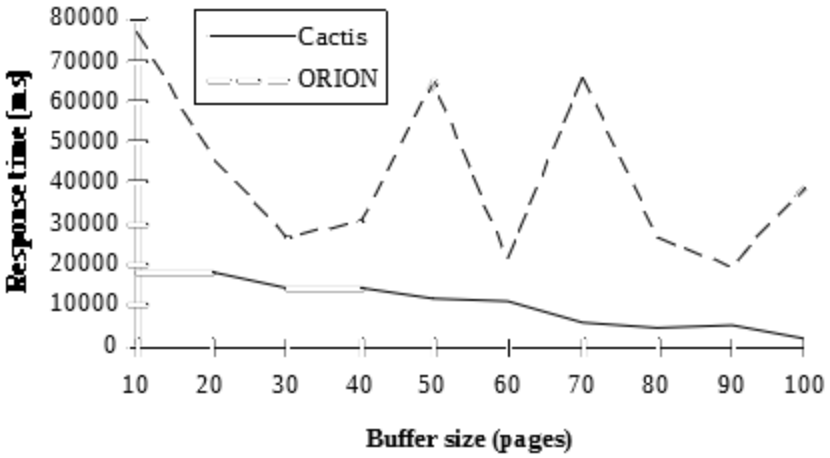

Figure 40a: Response time function of buffer size

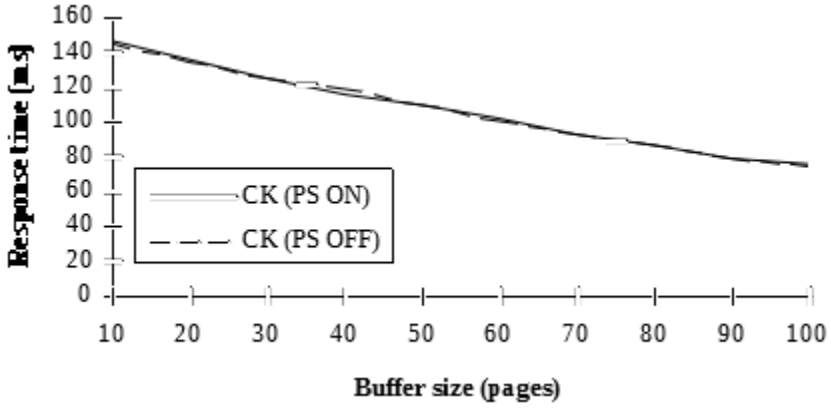

Figure 40b: Response time function of buffer size

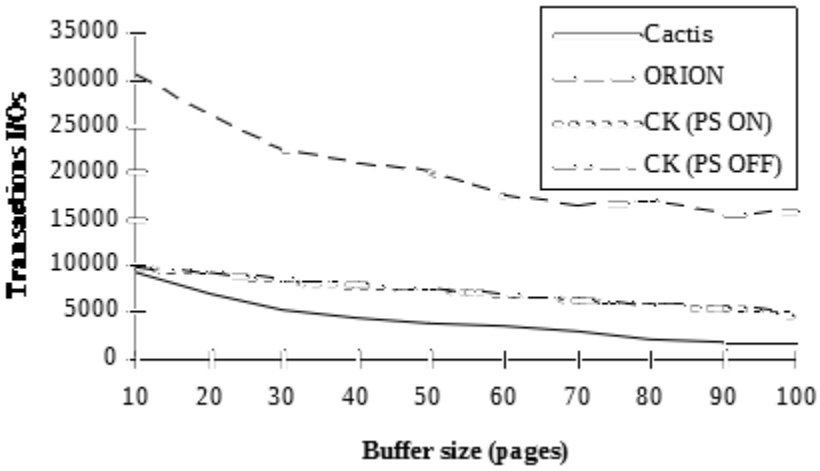

Figure 41: Transactions I/Os function of buffer size

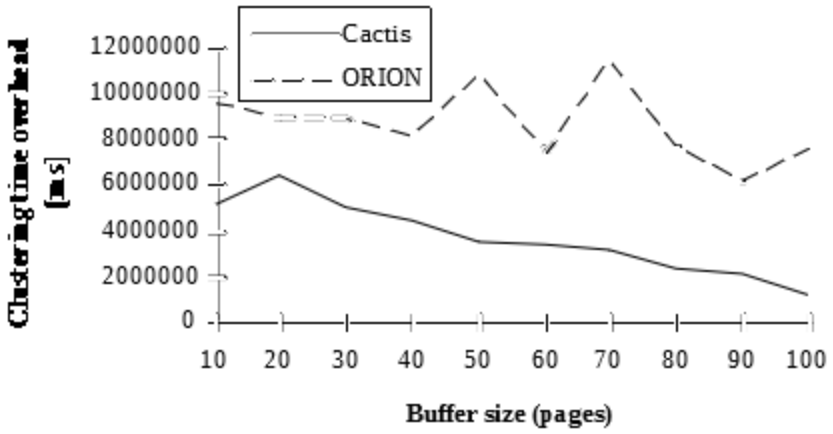

Figure 42a: Clustering time overhead function of buffer size

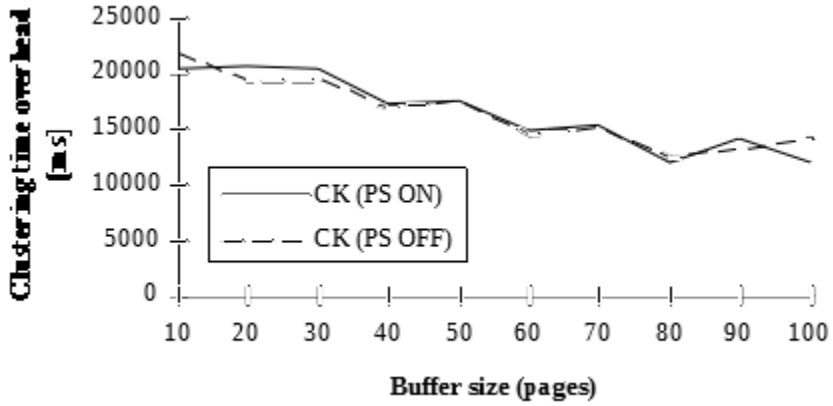

Figure 42b: Clustering time overhead function of buffer size

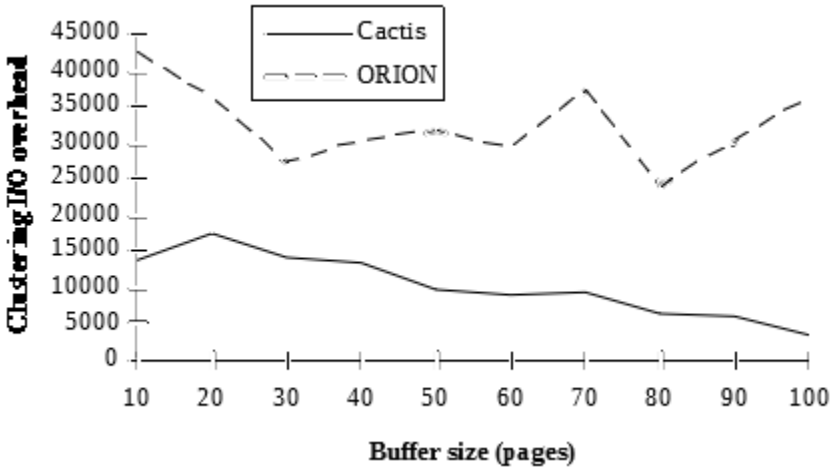

Figure 43a: Clustering I/O overhead function of buffer size

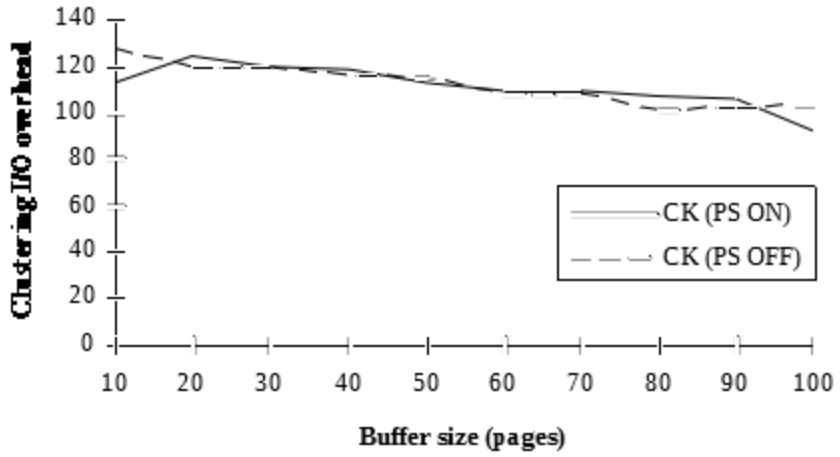

Figure 43b: Clustering I/O overhead function of buffer size

The database size should not vary when modifying the buffer capacity. This is true according to figure 44. Variations shown in the Cactis case are only due to the random factor inherent to simulation.

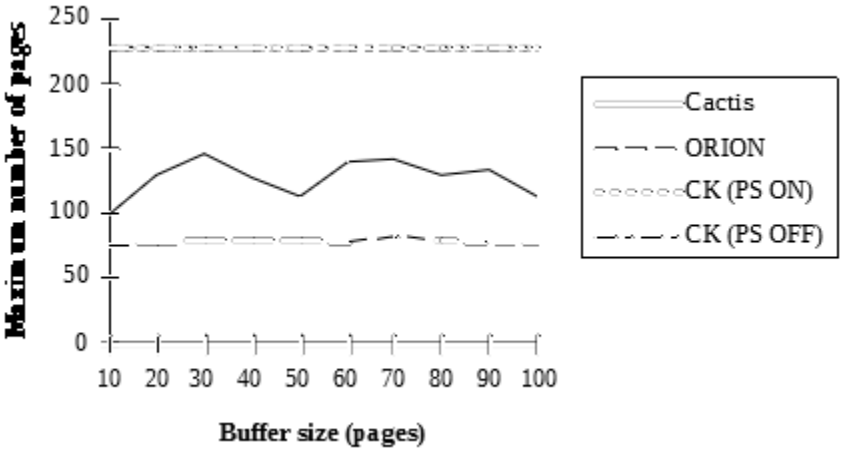

Figure 44: Maximum number of pages function of buffer size

System throughput (cf. figure 45) stays very close to optimal for Cactis and CK. For ORION, it increases fast then stays almost constant before decreasing for high numbers of objects, matching the response time (cf. figure 40a).

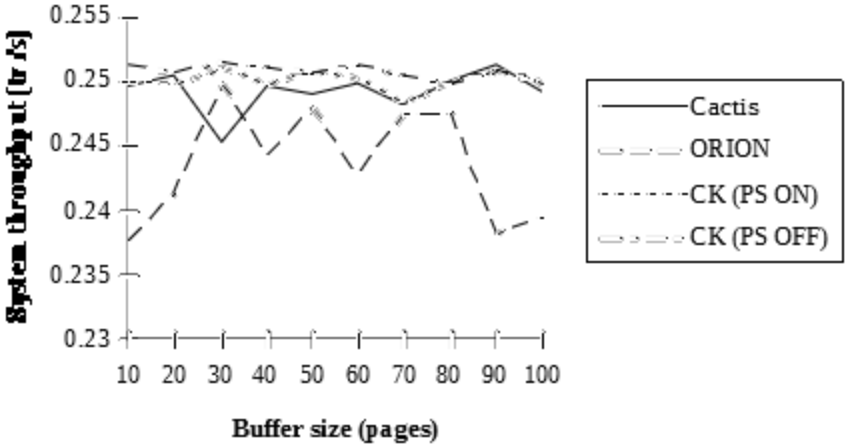

Figure 45: System throughput function of buffer size

3/ Effect of the Read/Write ratio

Read/Write ratio is an important factor when seeking to evaluate DBMSs performances. Furthermore, [CHANG89a] claims that the CK algorithm performs better when the Read/Write ratio is high. Thus we decided to study the effects of varying the Read/Write ratio. For our simulation experiments, we used an initial database of 400 objects and a buffer size of 10 pages. Table 6 gives the transaction probabilities we used, depending on the Read Percentage. What we call the Read Percentage is the cumulated probability of all the read transactions (i.e., transactions number 1 to 12 as they are defined in the Query Generation section).

		Read Percentage		
	78%	**60%**	**42%**	**24%**
PT1-PT12	0.065	0.05	0.035	0.02
PT13	0.1695 (Cactis) 0.169 (ORION) 0.17 (CK)	0.2995 (Cactis) 0.299 (ORION) 0.3 (CK)	0.4295 (Cactis) 0.429 (ORION) 0.43 (CK)	0.5595 (Cactis) 0.559 (ORION) 0.56 (CK)
PT14	0.05	0.1	0.15	0.2
PT15	0.0005 (Cactis) 0.001 (ORION) 0 (CK)	0.0005 (Cactis) 0.001 (ORION) 0 (CK)	0.0005 (Cactis) 0.001 (ORION) 0 (CK)	0.0005 (Cactis) 0.001 (ORION) 0 (CK)

Table 6: Transaction probabilities function of Read Percentage

The Cactis and ORION algorithms see their performance decrease when the Read Percentage decreases (cf. figure 46a). On the contrary, response time decreases along with the Read percentage in the case of CK (cf. figure 46b).

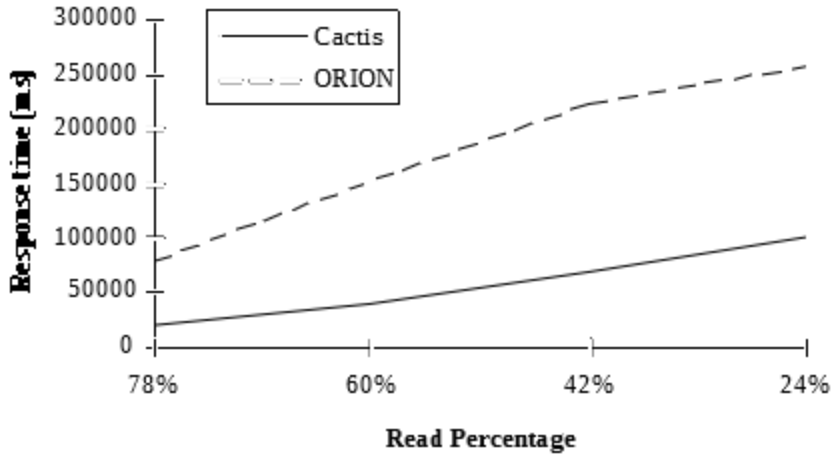

Figure 46a: Response time function of Read Percentage

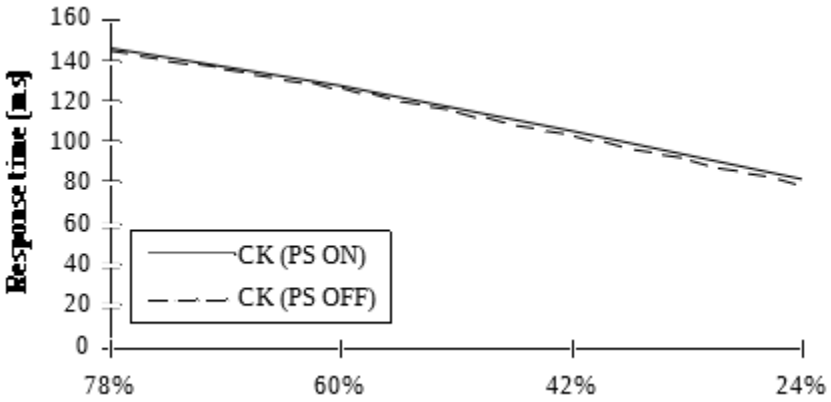

Figure 46b: Response time function of Read Percentage

Figures 48a, 48b and 49a, 49b show that clustering overhead increases for all the algorithms (confirming what is said in [CHAN89a]). Since Object Creation is a write operation, the more the Read Percentage drops, the more the database size increases, thus implying more clustering overhead, as shown in section 1/.

Parallely, transactions I/Os are slowly decreasing in number for Cactis and CK (cf. figure 47). This is because one single Object Creation is less costly than, for instance, such read transactions as Sequential Scans or Range Lookups. That explains the raise in performance for CK, since transactions I/Os drops from 10,000 to 5000 while clustering I/O overhead only rises from 100 to 500. In the Cactis case, clustering overhead is too important to compensate the decrease in transactions I/Os. For ORION, transactions I/Os

increase anyway because of the poor clustering ability of the algorithm. That explains the drop in system throughput shown by figure 51 for ORION.

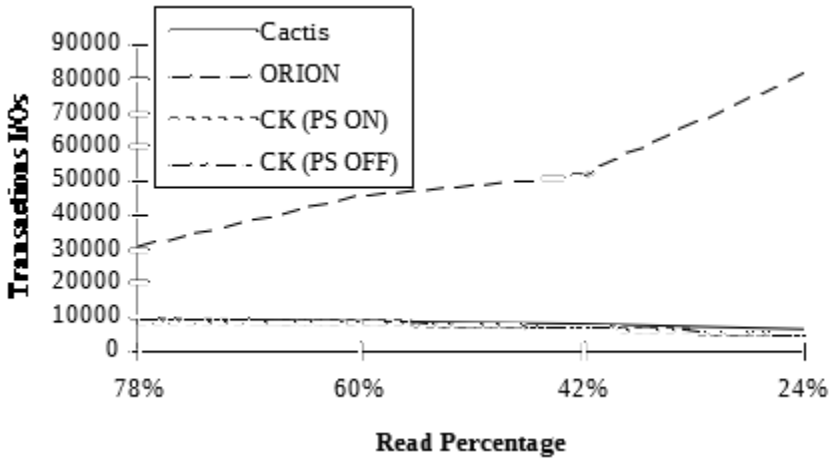

Figure 47: Transactions I/Os function of Read Percentage

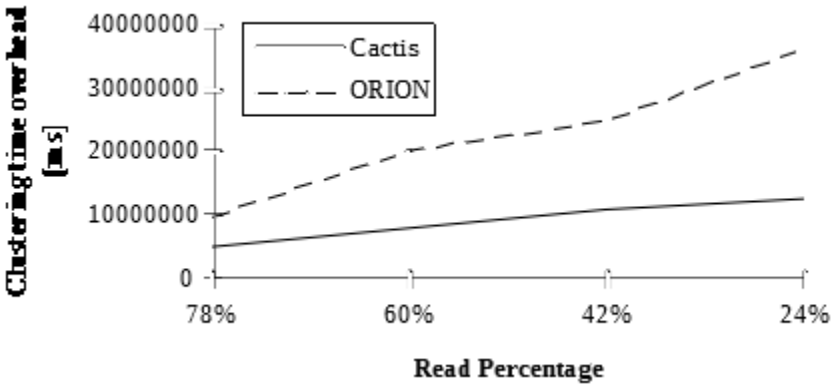

Figure 48a: Clustering time overhead function of Read Percentage

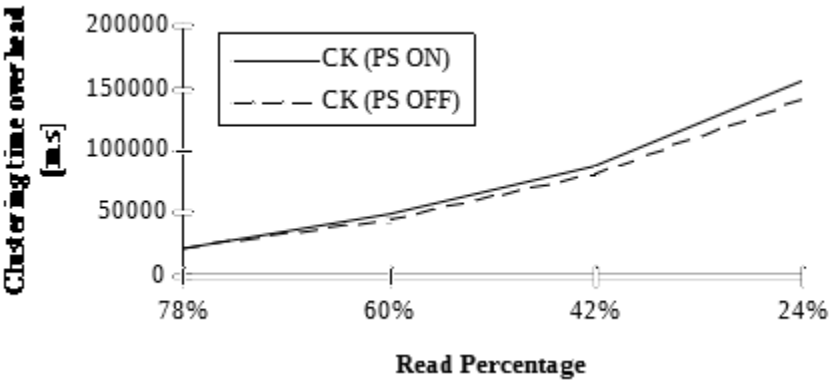

Figure 48b: Clustering time overhead function of Read Percentage

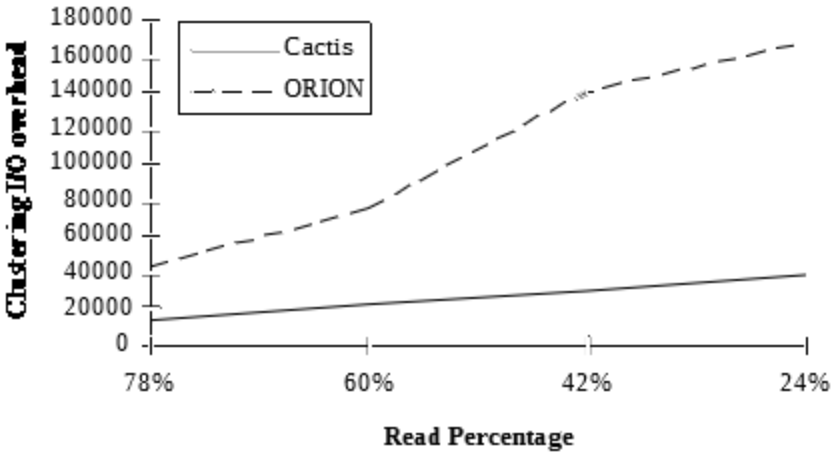

Figure 49a: Clustering I/O overhead function of Read Percentage

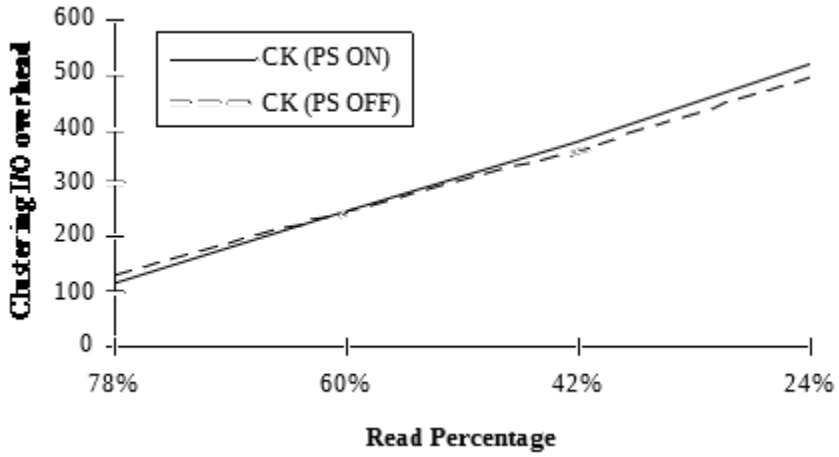

Figure 49b: Clustering I/O overhead function of Read Percentage

Number of pages (cf. figure 50) stays constant for the ORION and CK algorithms as it increases for Cactis. The reason is that ORION and CK place objects with their related objects and thus in possibly existing pages. On the contrary, when the Cactis algorithm reorganizes the database, pages are filled as the objects are clustered. Thus, an increase in database size always makes the number of disk pages used increase, which is not true for CK and ORION.

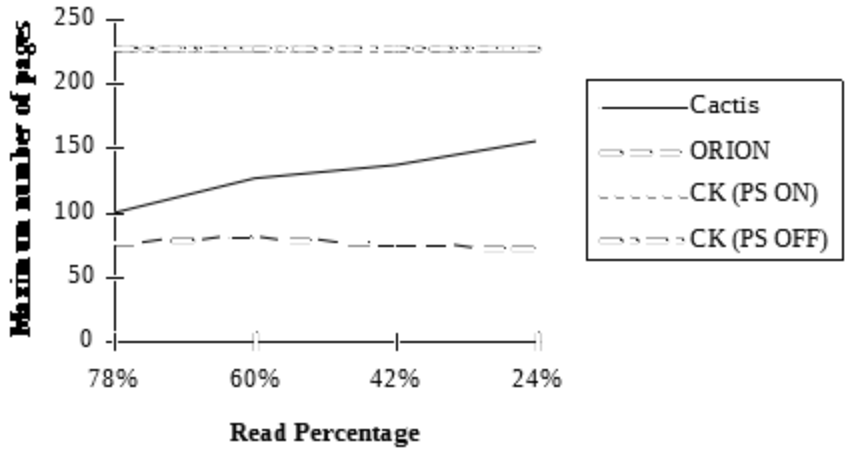

Figure 50: Maximum number of pages function of Read Percentage

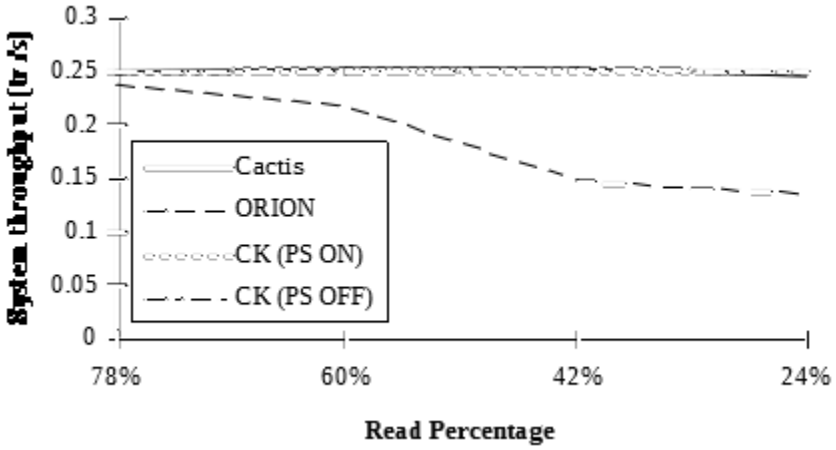

Figure 51: System throughput function of Read Percentage

III- Conclusions

• It is clear from our simulation experiments that the CK algorithm outperforms both Cactis and ORION in terms of overall performance. The results we obtained showed that this is due to both a good clustering capability and to the dynamic conception of the algorithm that allow an extremely low clustering overhead. Such a good behavior is achieved because the CK algorithm is activated only at object creation time and only accesses the few objects that are related to the newly created object once. Therefore, transactions are never blocked very long during clustering, as they are when the Cactis or the ORION algorithm is used. (The Cactis and ORION algorithms have to access all the objects in the database, even several times in the case of ORION, to

reorganize the database; and transactions cannot be run when a reorganization occurs.) CK good clustering capability is based on the users' hints that specify the inter-objects access frequencies for each structural relationship and thus allows to cluster together objects that are likable to be accessed together.

• Our simulations showed too that Cactis had also a good clustering capability. This is due to the use of statistics (i.e., objects access frequencies and relationships use frequencies) that allow to cluster together objects that are actually accessed together. Though, the Cactis algorithm is still completely outperformed by the CK algorithm. This is because, when using Cactis, clustering overhead increases very quickly with the number of objects, thus annihilating any gain achieved from good clustering capability. However, we have to keep in mind that this algorithm has been designed to run when the database is idle so that reclustering does not alter the database performance. Hence, if clustering overhead was not taken into account, the Cactis algorithm should perform about as well as the CK algorithm as long as the statistics used during the last reorganization are pertinent.

• In terms of disk space, the more a clustering algorithm is simple, the less space it should use. Actually, the more a clustering algorithm is complex, the more it clusters objects according to sharp criteria. Thus, a smaller number of objects are likely to be clustered in the same clustering unit (either a page or a segment). So the number of pages needed to store the database is greater. Our simulation experiments confirm that fact. The ORION algorithm is the less greedy algorithm in terms of disk pages used. Then the Cactis algorithm follows, using almost half the number of disk pages needed by CK to cluster the database. However, when reorganizing the database, the Cactis and ORION algorithms need to build a new set of pages before deleting the old one. Thus they require about twice as much space as our graphs show.

Conclusion

During this internship, I have gained knowledge in several areas. First I have improved my knowledge in the field of databases in general and in OODBs in particular, both by attending Dr. Gruenwald's classes and by performing research in the library. I now know much more about implementation issues such as clustering, buffering and versioning. I have also gained proficiency in the use of SLAM II simulation language.

The main problem I encountered while performing this study was the design of the simulation model. Papers I was lead to read often used simulation as an evaluation method, but very few of them gave a detailed simulation model that could have been a starting base, since it was not the aim of the papers. However, once the conceptual model had been designed, translating it into a SLAM II simulation model has been a minor issue.

Simulation experiments we performed showed that the Cactis algorithm is better than the ORION algorithm due to its good clustering capability and that the CK algorithm totally outperforms both other algorithms in terms of overall performance because it not only has a good clustering capability but also allows a very low clustering overhead.

The initial planning has been achieved for the most part. Though, we underestimated simulation time (that ranged from half an hour to nine hours, depending on the clustering algorithm used and the number of objects) and were not able to perform all the simulation experiments we wanted to.

Future research about this subject could be in a first step completing the simulation tests, notably by determining the effects of database changes, e.g., varying the probability for a class to belong to a composite class hierarchy, the average number and size of attributes, etc. Further research about page splitting in CK could also be done, including a more accurate performance evaluation of the two page splitting policies.

Then the next step would be the design and evaluation of one or several new clustering algorithms. One alternative would be to build a dynamic clustering algorithm that would use the same statistics as Cactis (i.e., objects access frequencies and relationships use frequencies) to cluster objects together, but could be able to use them at run-time (e.g., an

algorithm activated at object creation time, like CK). Such an algorithm could have Cactis' good clustering capability without being handicapped by an important clustering overhead.

Another alternative would be to modify the CK algorithm so that it does not use users' hints for inter-objects access frequencies any more and rely only on statistics, in order to make the algorithm even more accurate and performant. The problem with users' hints is that their accuracy depends the user (either the database administrator or a programmer) knowledge of the database. On the other hand, automatically gathered statistics show an exact image of the database status. Thus, by computing inter-objects access frequencies for each structural relationship and each object at run-time, a better performance should be achieved.

Bibliography

[ANDERSON90]
T.L. Anderson, A.J. Berre, M. Mallison, H.H. Porter III, B. Scheider
The HyperModel Benchmark
International Conference on Extending Database Technology, Venice, Italy, March 1990, Pages 317-331

[ANDREWS91a]
T. Andrews
Programming with Vbase
In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, Pages 130-177

[ANDREWS91b]
T. Andrews, C. Harris, K. Sinkel
ONTOS: A Persistent Database for C++
In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, Pages 387-406

[ATKINSON92]
M.P. Atkinson, A. Birnie, N. Jackson, P.C. Philbrow
Measuring Persistent Object Systems
5th International Workshop on Persistent Object Systems, San Miniato (Pisa), Italy, September 1992, Pages 63-85

[BANCILHON88]
F. Bancilhon, G. Bardebette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, F. Velez
The Design and Implementation of O_2, an Object-Oriented Database System
2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, Pages 1-22

[BANCILHON92]
F. Bancilhon, C. Delobel, P. Kanellakis
Building an Object-Oriented Database System: The Story of O_2
Morgan Kaufmann Publishers, 1992

[BANERJEE87]
J. Banerjee, H.-T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, H.-J. Kim
Data Model Issues for Object-Oriented Applications
ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, Pages 3-26

[BATORY85]
D.S. Batory, W. Kim
Modeling Concepts for VLSI CAD Objects
ACM Transactions on Database Systems, Vol. 10, No. 3, September 1985, Pages 322-346

[BENZAKEN90a]
V. Benzaken, C. Delobel
Enhancing Performance in a Persistent Object Store: Clustering Strategies in O_2
4th International Workshop on Persistent Object Systems, September 1990, Pages 403-412

[BENZAKEN90b]
V. Benzaken
An Evaluation Model for Clustering Strategies in the O_2 Object-Oriented Database System
3rd International Conference on Database Theory, Paris, France, December 1990, Pages 126-140

[BERRE91]
A.J. Berre, T.L. Anderson
The HyperModel Benchmark for Evaluating Object-Oriented Databases
In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, Pages 75-91

[BERTINO91]
E. Bertino, L. Martino
Object-Oriented Databases Management Systems: Concepts and Issues
IEEE Computer, April 1991, Pages 33-47

[CATTELL88]
R.G.G. Cattell
Object-Oriented DBMS Performance Measurement
2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, Pages 364-367

[CATTELL91a]
R.G.G. Cattell
Object Data Management: Object-Oriented and Extended Relational Database Systems
Addison-Wesley Publishing Company, 1991

[CATTELL91b]
R.G.G. Cattell
An Engineering Database Benchmark
In "The Benchmark Handbook for Database Transaction Processing Systems", Edited by Jim Gray, Morgan Kaufmann Publishers, 1991, Pages 247-281

[CHABRIDON92]
S. Chabridon, J.-C. Liao, Y. Ma, L. Gruenwald
Storage Management Techniques for Object-Oriented Database Systems
University of Oklahoma, School of Computer Science, Technical Report, December 1992

[CHABRIDON93]
S. Chabridon, J.-C. Liao, Y. Ma, L. Gruenwald
Clustering Techniques for Object-Oriented Database Systems
38th IEEE Computer Society International Conference, February 1993, San Francisco, Pages 232-242

[CHANG89a]
E.E. Chang
Effective Clustering and Buffering in an Object-Oriented DBMS
University of California, Berkeley, Computer Science Division (EECS), Technical Report No. UCB/CSD 89/515, June 1989

[CHANG89b]
E.E. Chang, R.H. Katz
Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS
ACM SIGMOD International Conference on Management of Data, Portland, Oregon, June 1989, Pages 348-357

[CHANG90]
E.E. Chang, R.H. Katz
Inheritance in computer-aided design databases: semantics and implementation issues
CAD, Vol. 22, No. 8, October 1990, Pages 489-499

[CHENG91]
J.R. Cheng, A.R. Hurson
Effective clustering of complex objects in object-oriented databases
ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, Pages 22-31

[DARMONT94]
J. Darmont
Comparaison de trois méthodes de groupement d'enregistrements (clustering) pour des bases de données orientées-objet en termes de temps de réponse et d'occupation disque
CUST, Institut des Sciences de L'Ingénieur, Blaise Pascal University, Clermont-Ferrand, France, Technical report, June 1994

[DEUX90]
O. DEUX et al.
The Story of O_2
IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, Pages 91-108

[DREW90]
P. Drew, R. King, S. Hudson
The Performance and Utility of the Cactis Implementation Algorithms
16th International Conference on Very Large Data Bases, Brisbane, Australia, August 1990, Pages 135-147

[FORD88]
S. Ford, J. Joseph, D.E. Langworthy, D.F. Lively, G. Pathak, E.R. Perez,
R.W. Peterson, D.M. Sparacin, S.M. Thatte, D.L. Wells, S. Agarwala
ZEITGEIST: Database Support for Object-Oriented Programming
2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, Pages 23-42

[GRUENWALD91]
L. Gruenwald, M.H. Eich
MMDB Reload Algorithms
ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, Pages 397-405

[HE93]
M. He, A.R. Hurson, L.L. Miller, D. Sheth
An Efficient Storage Protocol for Distributed Object-Oriented Databases
IEEE Parallel & Distributed Processing, 1993, Pages 606-610

[HUDSON89]
S.E. Hudson, R. King
Cactis: A Self-Adaptive Concurrent Implementation of an Object-Oriented Database Management System
ACM Transactions on Database Systems, Vol. 14, No. 3, September 1989, Pages 291-321

[HUDSON91]
S.E. Hudson, R. King
The Efficient Support of Functionally-Defined Data in Cactis
In "On Object-Oriented Database Systems", Edited by K.R. Dittrich, U. Doyal and A.P. Buchman, Springer-Verlag Topics in Information Systems, 1991, Pages 341-356

[HURSON93]
A.R. Hurson, S.H. Pakzad, J.-b. Cheng
Object-Oriented Database Management Systems: Evolution and Performance Issues
IEEE Computer, February 1993, Pages 48-60

[KATZ91]
R.H. Katz, E.E. Chang
Inheritance Issues in Computer-Aided Design Databases
In "On Object-Oriented Database Systems", Edited by K.R. Dittrich, U. Doyal and A.P. Buchman, Springer-Verlag Topics in Information Systems, 1991, Pages 45-52

[KHOSHAFIAN90]
S. Khoshafian, R. Abnous
Object Orientation
John Wiley & Sons, 1990

[KIM88]
W. Kim, H.-T. Chou
Versions of Schema for Object-Oriented Databases
14th International Conference on Very Large Data Bases, Los Angeles, USA, August 1988, Pages 148-159

[KIM90a]
W. Kim, J.F. Garza, N. Ballou, D. Woelk
Architecture of the ORION Next-Generation Database System
IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, Pages 109-124

[KIM90b]
W. Kim
Object-Oriented Databases: Definition and Research Directions
IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 3, September 1990, Pages 327-341

[MAIER86]
D. Maier, J. Stein, A. Otis, A. Purdy
Development of an Object-Oriented DBMS
ACM OOPSLA '86 Proceedings, September 1986, Pages 472-482

[MALHOTRA92]
A. Malhotra, K.J. Perry
Allocating Objects to Pages
5th International Workshop on Persistent Object Systems, San Miniato (Pisa), Italy, September 1992, Pages 3-10

[MONK92]
S.R. Monk, I. Sommerville
A model for versioning of classes in object-oriented databases
10th British National Conference on Databases, BNCOD 10, Aberdeen, Scotland, July 1992, Pages 42-58

[PRITSKER86]
A.A.B. Pritsker
Introduction to Simulation and SLAM II
Hasted Press (John Wiley & Sons), System Publishing Corporation, 1986

[SLAM86]
SLAM II Pocket Guide
© 1986 Pritsker & Associates

[SLAM92]
SLAM II Quick Reference Manual
© 1990, 1992 Pritsker Corporation

[SRINIVASAN91]
V. Srinivasan, M.J. Carey
Performance of B-Tree Concurrency Control Algorithms
ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, Pages 416-425

[TSANGARIS91]
M.M. Tsangaris, J.F. Naughton
A Stochastic Approach for Clustering in Object Bases
ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, Pages 12-21

[TSANGARIS92a]
M.M. Tsangaris, J.F. Naughton
On the Performance of Object Clustering Techniques
ACM SIGMOD International Conference on Management of Data, San Diego, California, June 1992, Pages 144-153

[TSANGARIS92b]
M.M. Tsangaris
Principles for Static Clustering for Object Oriented Databases
University of Wisconsin-Madison, Computer Science Department, Technical Report #1104, August 1992

[WILKES88]
W. Wilkes
Instance Inheritance Mechanisms for Object Oriented Databases
2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, Pages 274-279

[WU93]
Z. Wu, R. Leahy
An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation
IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, No. 11, November 1993, Pages 1101-1111

Appendix: Paper extracted from the Study

See pages 93 to 105.

CLUSTERING ALGORITHMS FOR OBJECT-ORIENTED DATABASES

Jérôme Darmont

Blaise Pascal University
Institut des Sciences de l'Ingénieur
Clermont-Ferrand, France

Le Gruenwald

University of Oklahoma
School of Computer Science
Norman, Oklahoma 73019
gruenwal@mailhost.ecn.uoknor.edu

Abstract: *It is widely acknowledged that good object clustering is critical to the performance of OODBs. Clustering means storing related objects close together on secondary storage so that when one object is accessed from disk, all its related objects are also brought into memory. Then access to these related objects is a main memory access that is much faster than a disk access. The aim of this paper is to compare the performance of three clustering algorithms: Cactis, CK and ORION. Simulation experiments we performed showed that the Cactis algorithm is better than the ORION algorithm and that the CK algorithm totally outperforms both other algorithms in terms of response time and clustering overhead.*

Keywords: Cactis, CK, Clustering, Object-Oriented Databases, ORION, Simulation

1. INTRODUCTION

There are several ways to improve response time (i.e., to limit the number of disk Input/Output) in a DBMS. Indexing, clustering (i.e., storing related entities close together on secondary storage) and buffering (i.e., fetching clustered entities at the same time and setting up replacement strategies) are widely used techniques in conventional DBMSs. However, OODBs present additional semantics like structural properties (inheritance, composite objects) and interrelationships between objects. New techniques have then to be thought of.

We have chosen to study three clustering algorithms found in the literature that we consider to be different enough to be representative of the current research on clustering techniques in OODBs: Cactis, CK and ORION clustering algorithms. The Cactis and ORION clustering algorithms are already implemented in DBMSs.

These particular algorithms have been selected because they present characteristics that are interesting to compare. For instance, CK and ORION are dynamic clustering algorithms as the Cactis clustering algorithm is static. ORION also uses only users' hints to cluster a database; the Cactis clustering algorithm uses only statistics about the database and the CK algorithm makes use of both.

Furthermore, the aim of previous performance evaluations performed on these algorithms was only to compare the effects of one particular clustering strategy to those of a "no clustering" policy [CHAN89a, HURS89]. We intend to compare each of these three algorithms to each other to determine which one performs the best in a given environment. The characteristics that make this algorithm the best should be isolated.

This paper is organized as follows. Section 2 explains the principles of clustering in OODBs. The three studied clustering algorithms are described in Section 3. Section 4 describes our simulation model. In Section 5, the simulation results are analyzed. Section 6 concludes this paper and provides future research directions.

2. CLUSTERING IN OODBs

2.1. Clustering principles

The goal of object clustering is to reduce the number of disk I/Os for object retrieval. Typically, the unit of data transferred from disk is a page instead of an individual object. If two objects are clustered on the same page, it will take only one disk I/O to access both objects successively. [HURS93]

Clustering algorithms attempt to improve the performance of object-oriented database systems by placing on the same page related sets of objects [TSAN92]. In object-oriented databases, complex objects are the basic units of data manipulation. The subobjects of a complex object may come from different classes. Traditional storage systems tend to group records of the same type physically close to each other on disk. This results in tedious and expensive reconstruction procedures (such as join operations) to retrieve complex objects. Therefore, it is logical to cluster related objects

of different classes together to achieve acceptable performance. [HURS93]

The problem of clustering can be seen as a graph partitioning problem. The nodes of the graph are the objects and the edges are the links between objects. This problem is NP-complete. However, as the graph of objects represents the database state, all is needed is an incremental solution where new objects are placed at the "right place". Most of the algorithms used can be classified as greedy algorithms: they scan the objects according to their links and try to place them into the same cluster unit. Thus the cost of clustering has no major impact on the overall system. [BENZ90]

2.2. Clustering strategies

According to [CATT91], clustering in an OODB can actually be performed in many different ways:

- *composite objects:* objects can be clustered according to aggregation relationships;
- *references:* some OODBs allow objects to be clustered according to relationships with other objects; composite objects clustering is, in fact, a special case of this, clustered by aggregation relationships;
- *object types:* objects may also be clustered by their types; if there is a generalization hierarchy, subtype instances may also be clustered in the same segment;
- *indexes:* as in relational DBMSs, it may be possible to cluster objects by an index on their attributes;
- *custom:* some OODBs allow clustering to be performed "on the fly".

Unless objects are stored redundantly, an object can generally be clustered according to one of these rules. Where the rules do not conflict, however, it is possible to follow multiple clustering rules.

Clustering may be performed at two levels:

- *pages:* objects may be clustered according to the smallest physical unit read from disk, which is normally a page; this type of clustering can produce the greatest gains in performance when a "working set" of objects cannot be precisely defined for all applications; page clustering is more useful for clustering by index, reference and composite objects;
- *segments:* objects may be clustered in larger units, when the user is able to specify a meaningful logical grouping for segmentation; segment clustering is most useful for type clustering; it may also be used for composite objects, if used at a sufficiently course grain.

The largest performance gains are generally afforded by page clustering, since pages are the unit of access from disk and a "working set" of pages is selected dynamically according to the access characteristics of an application program. Segment clustering produces efficiency gains only if relatively large contiguous units are transferred from disk, or when efficiency gains can be made through grouping operations (for example, for composite objects deletion).

2.3. Users' hints

To expedite the retrieval of related data, database systems often take hints from the user (or database administrator) to store related data physically close together [KIM90b]. For example, the GemStone database administrator, or a savvy application programmer can hint GemStone that certain objects are often used together and so should be clustered on disk [MAIE86]. The VBASE system allows explicit clustering hints when objects are created [ANDR91a]. The strategy adopted in ONTOS is to allow the programmer to specify clustering and to provide tools for reclustering when more experience with the applications permits better choices to be made [ANDR91b].

2.4. Static versus dynamic clustering

In the *static* case, clustering is done at the time objects are created and no reorganization is implied when the links between objects are updated [BENZ90]. A static clustering scheme offers a good placement policy for complex objects but does not take into account the dynamic evolution of objects. In applications such as design databases, objects are constantly updated during early parts of the design cycle. Frequent updates may destroy the initially clustered structure. To keep the object structure optimized, reorganization might be necessary for efficient future accesses [DEUX90].

Dynamic clustering is done at run time when objects are accessed concurrently and becomes attractive in an environment where the read operations dominate the write operations [BENZ90]. A dynamic clustering scheme should try to recluster when scattered access cost becomes too high. However, reclustering will generate overhead such as extra disk I/Os, so it is important to determine when a reorganization should occur. If the overhead is not justified, reclustering may actually degrade the overall performance [CHEN91].

3. CLUSTERING ALGORITHMS

3.1. Cactis clustering algorithm

Cactis [HUDS89] is an object-oriented, multi-user DBMS developed at the University of Colorado. It is designed to support applications that require rich data modeling capabilities and the ability to specify functionally-defined data.

The Cactis clustering algorithm is designed to place objects that are frequently referenced together into the same block (i.e., page, i.e., I/O unit) on secondary storage. It can improve response time up to 60%.

In order to improve the locality of data references, data is clustered on the basis of usage patterns. A count of the total number of times each object in the database is accessed is kept, as well as the number of times each relationship between objects in the process of attribute evaluation or marking out-of-date is crossed. Then, the database is periodically reorganized on the basis of this information. The database is packed into blocks using the greedy algorithm shown in Figure 1.

This clustering algorithm is also implemented in the Zeitgeist system [FORD88].

The Cactis clustering algorithm is a static algorithm since it is periodically used to recluster the database when the database is idle. This implies that the database is not clustered on the first run because no information about the database is available [CHAB93].

This algorithm does not require users' hints. This is an advantage since no arbitrary choice has to be made by the user [CHAB93]. But it also implies some time overhead (time to compute total number of times each object is accessed and number of times each relationship is crossed) and space overhead (the main memory space used to store the counters grows with the database size). It also raises the problem of getting pertinent statistics about the database.

Repeat
    Choose the most referenced object in the database that has not yet been assigned a block.
    Place this object into a new block.
    Repeat
        Choose the relationship belonging to some object assigned to the block such that:
            (1) the relationship is connected to an unassigned object outside the block and,
            (2) the total usage count for the relationship is the highest.
        Assign the object attached to this relationship to the block.
    Until the block is full.
Until all objects are assigned blocks.

Figure 1: Cactis clustering algorithm [HUDS89]

3.2. ORION clustering method

ORION is a series of next-generation database systems that have been prototyped at MCC (Microelectronics Computer Technology Corp.) as vehicles for research into the next-generation database architecture and into the integration of programming languages and databases [KIM90a]. ORION has been designed for Artificial Intelligence (AI), Computer-Aided Design and Manufacturing (CAD/CAM) and Office Information System (IOS) applications [BANE87].

ORION supports only a simple clustering scheme. Instances of the same class are clustered in the same physical segment (i.e., a number of blocks or pages). Each class is associated with one single segment. [KIM90a]

But ORION also provides direct support for *composite objects*, i.e., objects with a hierarchy of exclusive component objects (see Figure 2). The hierarchy of classes to which the objects belong is a *composite object hierarchy*. The object-oriented data model, in its conventional form, is sufficient to represent a collection of related objects. However, it does not capture the IS-PART-OF relationship between objects; one object simply references, but does not own, other objects. A composite object hierarchy captures the IS-PART-OF relationship between a parent class and its component classes, whereas a class hierarchy represents the IS-A relationship between a superclass and its subclasses. [BANE87]

Then it becomes advantageous to store instances of multiple classes in the same segment. User assistance is required to determine which classes should share the segment. The user can dynamically issue a Cluster message containing a “ListOfClassNames” argument specifying the classes that are to be placed in the same segment. [BANE87]

In ORION, segments have a fixed size. So the number of pages they contain gives the number of I/Os necessary to load the segment. When a segment is full, a new page is allocated and linked to the segment (a pointer must be maintained in the segment descriptor). This

implies some overhead to find the address of each additional page [CHAB93].

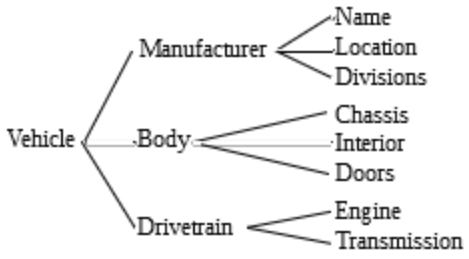

Figure 2: Example of composite object

The advantage of this method is its simplicity that makes the method fast and easy to implement since no cost model is defined and no overhead is implied to determine what is the optimal storage unit for an object. But simplicity also turns to a limitation since users' hints can only be based on the static information given by the data model and not on some information determined by the database usage and which could lead to a better clustering. [CHAB93]

3.3. CK clustering algorithm

The CK algorithm (from its authors' names: Chang and Katz) is defined in the CAD/CAM context. It can improve response time up to 200% when the Read/Write ratio is high (which is true for real CAD applications) [CHAN89b]. The CK algorithm makes use of several new concepts, such as structural relationships and instance-to-instance inheritance.

3.3.1. Structural relationships

Structural relationships are versions, configurations and equivalence relationships.

Objects sharing the same interface but having different implementations are called versions [BATO85]. They represent different design alternatives. For example, if an object is identified by the pattern: *Name[Version].Type* where "Name" is the object name, "Version" its version number and "Type" its type; Nice[1].car, Nice[2].car and Nice[3].car would be three versions of the same object "Nice" type of which is "car".

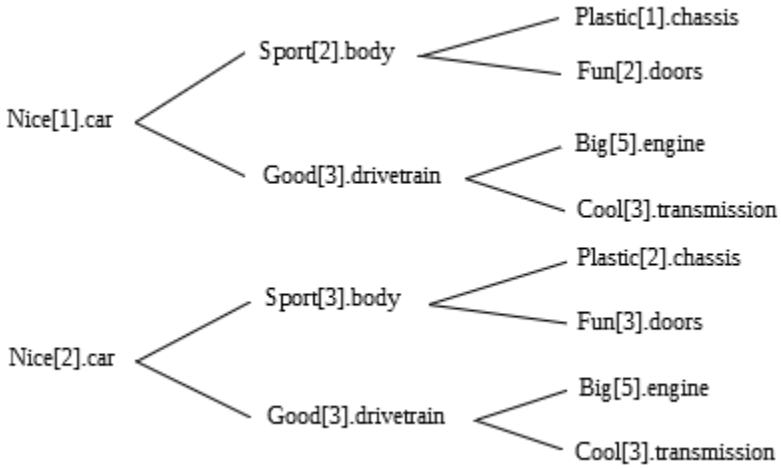

Figure 3: Example of configurations

A very important characteristic of OODBs is the presence of composite (complex or nested) objects. This concept is represented through composite/component relationships among objects. Coupling the concept of versions with composite objects leads to configurations. A configuration is a composite unit whose components are bound to specific versions (see Figure 3) [CHAN90].

If two objects are alternative representations of the same real world entity, they are equivalent.

3.3.2. Instance-to-instance inheritance

Besides structural relationships, inheritance provides additional semantics. As in object-oriented programming languages, a class/subclass hierarchy can be defined for an OODB based on the IS-A relationship. A subclass inherits the structure (i.e., attributes' definitions) and the methods of its superclass. However, in OODBs, this form of inheritance (called type inheritance) is not sufficient. [CHAB93]

The CK algorithm also uses instance-to-instance inheritance that not only transfers the existence of attributes from one object to another (like type inheritance), but moreover the values of these attributes [WILK88].

Instance-to-instance inheritance is important in computer-aided design databases, since a new version tends to resemble its immediate ancestor. It is useful if a new version can inherit its attributes' values, and more importantly its constraints, from its ancestor. [KATZ91]

3.3.3. Algorithm presentation

Instance-to-instance inheritance introduces more complexity because it allows attributes to be selectively inherited at run-time. This run-time flexibility requires a sophisticated approach for clustering. The CK algorithm is based on inter-objects access frequencies (given by the user at data type creation time) for each kind of structural relationship, e.g., 20% of access along version relationships, 75% of access along configuration relationships and 5% of access along equivalence relationships.

When a new object is created, the algorithm chooses an initial placement based on which relationship is most frequently used to reach the object (in the above example, a new instance would probably be placed in the same page as its composite objects). Then, for each inherited attribute, cost formulas are used to choose between implementation by copy or by reference, i.e., either by copying the attribute's value or reference it with a pointer. The augmented access frequencies (i.e., relationship traversal frequencies plus inheritance traversal frequencies) may change the initial placement. The clustering algorithm pseudo code is given in Figure 4.

Then, if the best candidate page is full, either the next best candidate page is chosen or the page is split if the expected access cost resulting from the split is an improvement over placement in the next best candidate page.

Page splitting is performed by a greedy algorithm that partitions the inheritance-dependency graph into two sub-graphs that each fit into one page. This algorithm is not optimal, but it is linear (whereas an exact partitioning algorithm would be NP-complete). It is described in Figure 5.

4. SIMULATION MODEL

4.1. Object base

For our simulations, we used a random object base whose class hierarchy forms a DAG, as in [HE93]. The database generation was performed in two phases: first generate class hierarchies and class definition (see Figure 6), then generate instances for these classes. To simplify the class hierarchy, we did not take into account multiple inheritance because it has no effect on clustering. We also assumed that a given class had one single ancestor version and one single descendant version but could have several component classes or equivalent classes.

Instance creation has been designed as a special kind of query. However, the initial database is to be created before any other query can occur, given an initial number of objects. The method we used to generate instances is shown in Figure 7.

4.2. Query generation

The HyperModel Benchmark [ANDE90, BERR91] provides 20 different types of transactions. From those 20, we have isolated 15 types of transactions (some of them are slightly modified to match the structural relationships we use). Each transaction has a probability to occur.

- *Name Lookup*: Retrieve a randomly selected object; fetch one of its (randomly selected) attributes' value.
- *Range Lookup*: Select a class at random; select one of its attributes at random; determine randomly two test values; fetch all the attributes of all the instances of the class whose selected attribute's value are in the range defined by the test values.
- *Group Lookup*: Given a randomly selected starting object, fetch all the attributes of either:
  - all its component objects,
  - all its equivalent objects,
  - all its descendant versions.
- *Reference Lookup*: Given a randomly selected starting object, fetch all the attributes of either:
  - its composite object,
  - all its ancestor versions.
- *Sequential Scan*: Select a class at random; select one of its attributes at random; fetch this attribute's values for every instance of the class.

```
PROCEDURE cluster_object(target_objet)
BEGIN
    /* step 1: get initial information */
    cluster_policy:=get_policy( );                  /* Is page splitting enabled? */
    copy_set:=get_by_copy_set( );                   /* Inherited attributes implemented by copy. */
    ref_set:=get_by_ref_set( );                     /* Inherited attributes implemented by reference. */
    inh_page_set:=get_all_inh_page( );              /* Source pages for inherited attributes. */
    struct_page_set:=get_all_struct_page( );        /* Source pages for structural objects. */
    page_set:=inh_page_set+struct_page_set;
    /* step 2: calculate ref_set lookup cost for each page */
    FOR p IN page_set                               /* If by-reference attribute r is */
        FOR r IN ref_set                            /* not in page p, storing target object */
            IF r NOT_IN p                           /* in page p requires one run-time */
            BEGIN                                   /* lookup for attribute r. */
                weight(p):=1/(prob(p,struct_rel));
                Ref_LookUp(p):=Ref_LookUp(p)+weight(p);
            END;
    /* step 3: calculate copy_set lookup and storage cost for each page */
    FOR c IN copy_set                               /* If by-copy attribute c is not in page */
        FOR p IN page_set                           /* p, we could either cache it in page p */
            IF c NOT_IN p                           /* or change its implementation to be */
            BEGIN                                   /* by-reference. */
                weight(p):=1/(prob(p,struct_rel));
                Copy_storage(p):=Copy_storage(p)+size_of(c);
                Copy_LookUp(p):=Copy_LookUp(p)+weight(p);
            END;
    /* step 4: calculate total cost of every page. If by-copy attributes are */
    /*             implemented by reference, the total cost of storing target object */
    /*             in page p is represented by Total(p,1). Otherwise, the cost */
    /*             is represented by Total(p,2). */
    FOR p IN page_set
        Total_cost(p,1):=Ref_LookUp(p)*Lookup_cost+Copy_LookUp(p)*Lookup_cost;
        Total_cost(p,2):=Ref_LookUp(p)*Lookup_cost+Copy_storage(p)*Storage_cost;
    /* step 5: pick up best candidate page and try to insert the object */
    candidate_page:=Minimum(Total_cost);
    IF (cluster_policy EQ no_split)
        WHILE (NOT_FIT(candidate_page))
            candidate_page:=Next_Min(Total_cost);
    IF ((cluster_policy EQ page_split) AND (NOT_FIT(candidate_page))
        Split_page(candidate_page);
END;
```

Figure 4: Pseudo code for CK clustering algorithm [CHAN90]

The Page_split algorithm assumes that the arc costs C_{ei} (i.e., run-time lookup cost) between objects are always maintained and sorted. The node capacity Cap_{Vi} (i.e., the object size) is also maintained. Subset A and B represent the sets of objects assigned to the new pages after splitting. Both subsets are empty ate the beginning. E is the initial set of arcs relating the objects.

- Step (1): Select the maximum value arc from E as e_{target} and set E to be (E - {e_{target}}). Let v_{head} and v_{tail} be the head and the tail nodes of e_{target}.
- Step (2): Supposed both v_{head} and v_{tail} are new to subsets A and B. Insert v_{head} and v_{tail} in subset A if Cap_{Vhead} plus Cap_{Vtail} is less than the remaining capacity of subset A. Otherwise, insert v_{head} and v_{tail} in subset B if subset B has space for these nodes. If neither subset A or B could accommodate both v_{head} and v_{tail}, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (3): Supposed v_{head} is in subset A and v_{tail} is not in subset A or B. Insert v_{tail} into subset A if feasible. Otherwise, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (4): Supposed both v_{head} and v_{tail} are visited before, a broken arc is found and $C_{etarget}$ is added into C_{total}.
- Step (5): Look back to step (1) until arc set E is empty.

Figure 5: Page_split algorithm [CHAN90]

Given a number of classes, we first build a class hierarchy that includes versions **(1)**. Then we build a composite hierarchy and add equivalence relationships **(2)**.

(1) A new class is added.
A random number of versions of this class is added (descendant versions).
If the new class has a superclass (given a probability of having a superclass) then
randomly select a superclass among the existing classes,
inherit attributes and methods of the superclass,
for each additional version of the class:
randomly select a superclass among the initial class superclass descendant versions,
inherit attributes and methods of the superclass.
Add additional random attributes and methods to all versions
(sizes of attributes and methods are assigned randomly).
Compute object size for these classes.

(2) Scan all the classes.
For each class:
If it is a component of one class (given a probability of being component) then
randomly select a class composed of the new class.
If it has an equivalent class (given a probability of having an equivalent) then
randomly select an equivalent class.

Figure 6: Class lattice generation

For each new object:
Randomly select a class.
If the new object class is a component of another class then
randomly select an instance of this class (if any) to be composed of the new object.
If the new object class is a version then
randomly select one ancestor object in the new object class ancestor class,
If using CK, inherit values of common attributes (either by copy or by reference).
If the new object class has an equivalent class then
randomly select one equivalent object among instances of the equivalent class.

Figure 7: Instances generation

• *Closure Traversal*: Given a randomly selected starting object, follow one of the three structural relationships (i.e., version, configuration or equivalence) to a certain predefined (random) depth D; fetch a random attribute from the resulting object; the followed relationship can be either always the same or randomly selected.
• *Editing*: Select an object at random; update one of its attribute (randomly chosen) with a random value.
• *Object Creation*: Creation of a new object (cf. object base generation). This activates the CK clustering algorithm.
• *Reclustering*: The ORION clustering algorithm needs a "Cluster message" to be dynamically activated [BANE87]. The Cactis clustering algorithm is static. We can assume it will also wait for a cluster message before reorganizing the database. However, cluster messages for the Cactis algorithm should be far less frequent than cluster messages for the ORION algorithm since the Cactis clustering algorithm is supposed to run when the database is idle [HUDS89].

4.3. Overall model

The overall simulation model is inspired by the one provided in [CHAN89a]. It is composed as follows (see Figure 8).
• Client module: After a predefined think time, the client issues the transactions to the Transaction Manager according to some frequencies of occurence.
• Transaction Manager module: The transaction manager extracts from transactions which objects have to be accessed or updated, and performs the operations. In the case of a regular operation, object requests are sent to the Buffering Manager. In the case of instance creation or a Cluster message, the Clustering Manager is invoked.
• Buffering Manager: The Buffering Manager checks if an object is in main memory and requests it to the I/O Subsystem if it is not. It also deals with page replacement strategie (when a new page is needed, the oldest page in memory is dropped and replaced by the new one).

• Clustering Manager: The Clustering Manager is activated depending on the algorithm (i.e., Cactis, CK or ORION) it implements. It deals with reorganizing the database on secondary storage to achieve better performance.
• I/O Subsystem: This module deals with physical accesses to secondary storage.

4.4. Simulation parameters

Tables 1 and 2 provide the simulation parameters we used for our simulation experiments.

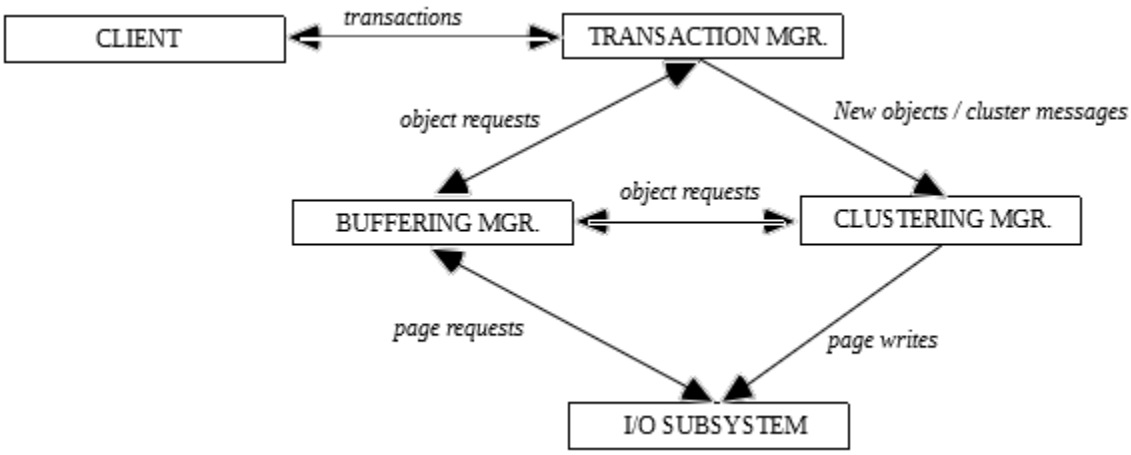

Figure 8: Overall simulation model

5. SIMULATION RESULTS

To compare the performance of the three clustering algorithms, we conducted four testing cases: varying the database size, the workload, the buffer capacity and the Read/Write ratio. Due to space limitation, we present in the following subsections only some of the results we obtained.

5.1. Effects of the database size

We first tested the effect of varying the initial number of objects in the database using a uniform random distribution to choose the transactions' starting objects. This is not always realistic since there may be objects that are "hotter" (i.e., more frequently accessed) than others in real world applications. Furthermore, the performance of the Cactis clustering algorithm depends on run-time computed statistics, such as object's access frequencies, that are not the same when using different random distributions to select the transactions starting object. So we implemented in a second series of simulations a normal random distribution for the transactions' starting objects, which is very similar to the "skewed random" function introduced in [TSAN92].

Figure 9a shows that response time when using Cactis is 24 % lower than when using ORION. Figure 9b shows that, for CK, response time increases linearly with the number of objects and that CK algorithm totally outperforms the other two (being about 800 times better than Cactis). "PS ON" and "PS OFF" stand for page splitting used or not.

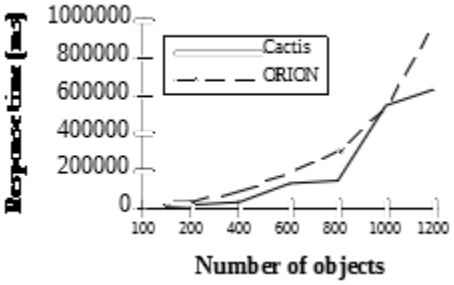

Figure 9a: Response time vs. number of objects (U)

Figures 9a and 10 show indeed that Cactis performs 1.5 times better when objects are not accessed through a uniform distribution,

especially for intermediate numbers of objects (between 400 and 600).

Parameter name	Designation	Value	Justification, References
RAVGTHINK	Average client think time	4 s	[CHAN89a]
RCC	Average locking/unlocking time (concurrency control)	0.5 ms	[SRIN91]
IMLVL	Multiprogramming level	10	[GRUE91]
IWDSIZE	Size of one memory word	4 bytes	[GRUE91]
ICPU	CPU power	2 Mips	[GRUE91]
RMACC	Memory word access time	0.0001 ms	[GRUE91]
RMTEST	Time for comparison of two memory words	0.0007 ms	Two memory accesses, one subtraction
IPGSIZE	Size of disk page	2048 bytes	[CHEN91]
RSEEK	Average disk seek time	28 ms	[CHEN91]
RLATENCY	Average disk latency time	8,33 ms	[CHEN91]
RTRANSFER	Disk page transfer time	1.28 ms	[CHEN91]

Table 1: Static parameters

Parameter name	Designation	Default value	Range
NCL	Number of classes	20	10-30
IAVGVER	Average number of versions per class	3	1-5
RPSUPER	Probability for a class of having a superclass	0.9	0-1
RPCOMP	Probability for a class of being a component class	0.5	0-1
RPEQUI	Probability for a class of having an equivalent class	0.1	0-1
INOBJ	Initial number of objects	400	100-1000
IAVGASIZE	Average attribute size	1 word	1-3 words
IAVGNATTR	Average number of attributes per class	10	5-20
IBUFF	Size of memory buffer	10 pages	10-100 pages
IMD	Maximum depth in Closure Traversals	5	3-10
ISEGSIZE	Default segment size (ORION)	5	3-10
ITHRESHOLD	Update Threshold (CK)	25	0-255
ISCALEF	Scale factor (CK)	0.5	0-1
ISPLIT	Page split policy (CK)	ON	ON/OFF
PT1-PT12	Probability of Read Transaction (#1-12)	0.065	0-1
PT13	Probability of Editing	0.1695 (Cactis) 0.169 (ORION) 0.17 (CK)	0-1
PT14	Probability of Object Creation	0.05	0-1
PT15	Probability of Reclustering	0.0005 (Cactis) 0.001 (ORION) 0 (CK)	0-1

Table 2: Dynamic parameters

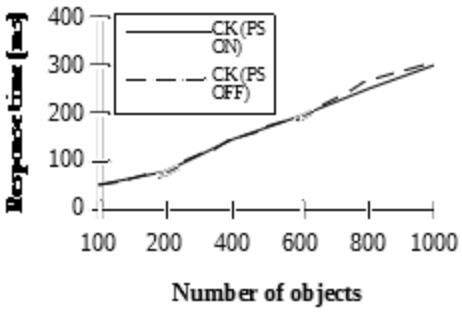

Figure 9b: Response time vs. number of objects (U)

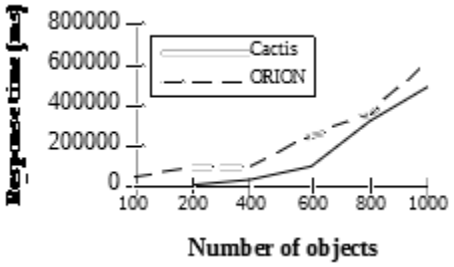

Figure 10: Response time vs. number of objects (N)

The more a clustering algorithm is complex (i.e., the more it clusters object according to precise rules), the more it uses a greater amount of disk pages to cluster the object base. The maximum number of disk pages used (as shown in Figure 11), as expected, is higher for the more complex algorithms, i.e., CK needs 1.8 times as many pages as Cactis and Cactis needs 1.3 times as many pages as ORION, for which number of pages increases linearly.

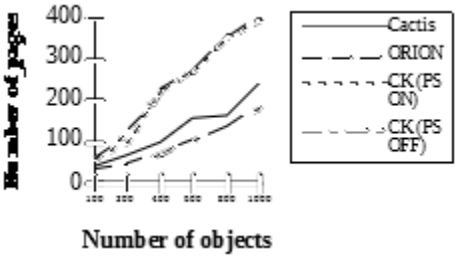

Figure 11: Number of pages vs. number of objects

5.2. Effect of the buffer capacity

On one hand, increasing the buffer capacity may lessen the effects of clustering since a given set of related objects has a higher probability of being in main memory instead of on secondary storage. On the other hand, objects are also accessed when reorganizing the database. Thus, increasing buffer capacity should decrease clustering overhead, especially for the ORION clustering algorithm that may make several non-consecutive accesses to each object. We performed this set of simulations using an initial database of 400 objects and selecting the transactions' starting objects from a uniform random distribution.

Figures 12a and 12b show that response time decrease linearly with the buffer size for the Cactis and CK algorithms. The dual effect of increasing the buffer capacity is seen on both transactions I/Os and clustering overhead. The ORION algorithm has a similar behavior, but the gain in performance is seen earlier than with the other algorithms and then the gain in performance is less important (see Figure 12a). We can explain this by the fact that the ORION algorithm uses a smaller amount of pages than the other algorithms to cluster the database. Thus, the buffer size grows faster relatively to the database size. For instance, a buffer size of 20 pages represents 25% of the database size for ORION versus only 15% for Cactis and 9% for CK. Figure 12a presents rather big variations in performance when buffer capacity grows over 40 pages. However, these values oscillate around a mean value that decreases slowly but linearly.

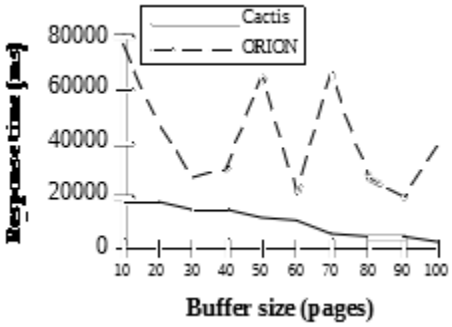

Figure 12a: Response time vs. buffer size

5.3. Effect of the Read/Write ratio

Read/Write ratio is an important factor when seeking to evaluate DBMSs performances. Furthermore, [CHAN89a] claims that CK

algorithm performs better when the Read/Write ratio is high. For our simulation experiments, we used an initial database of 400 objects and a buffer size of 10 pages.

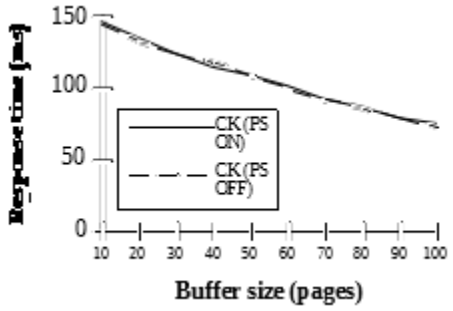

Figure 12b: Response time vs. buffer size

The performance of the Cactis and ORION algorithms decreases when the Read/Write ratio decreases (see Figure 13a). On the contrary, response time decreases along with the Read/Write ratio in the case of CK (see Figure 13b).

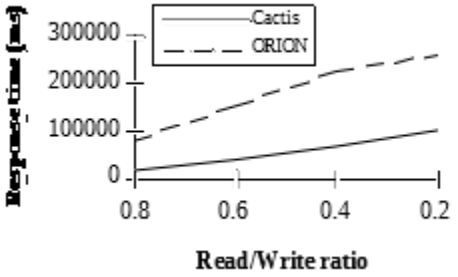

Figure 13a: Response time vs. R/W ratio

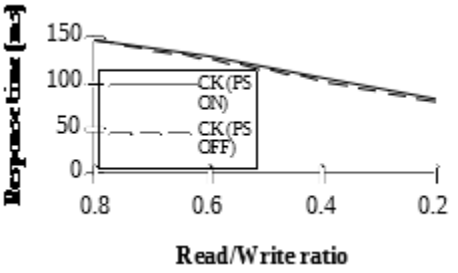

Figure 13b: Response time vs. R/W ratio

Since Object Creation is a write operation, the more the Read Percentage drops, the more the database size increases, thus implying more clustering overhead (confirming what is said in [CHAN89a]). Parallely, transactions I/Os are slowly decreasing in number for Cactis and CK. This is because one single Object Creation is less costly than, for instance, such read transactions as Sequential Scans or Range Lookups. That explains the raise in performance for CK, since transactions I/Os drops from 10,000 to 5000 while clustering I/O overhead only rises from 100 to 500. In the Cactis case, clustering overhead is too important to compensate the decrease in transactions I/Os. For ORION, transactions I/Os increase anyway because of the poor clustering ability of the algorithm.

6. CONCLUSIONS

It is clear from our simulation experiments that the CK algorithm outperforms both Cactis and ORION in terms of overall performance. The results we obtained showed that this is due to both a good clustering capability and to the dynamic conception of the algorithm that allow an extremely low clustering overhead. Such a good behavior is achieved because the CK algorithm is activated only at object creation time and only accesses the few objects that are related to the newly created object once. Therefore, transactions are never blocked very long during clustering, as they are when the Cactis or the ORION algorithm is used. (The Cactis and ORION algorithms have to access all the objects in the database, even several times in the case of ORION, to reorganize the database; and transactions cannot be run when a reorganization occurs.) CK good clustering capability is based on the users' hints that specify the inter-objects access frequencies for each structural relationship and thus allows to cluster together objects that are likable to be accessed together.

Our simulations showed too that Cactis had also a good clustering capability. This is due to the use of statistics (i.e., objects access frequencies and relationships use frequencies) that allow to cluster together objects that are actually accessed together. Though, the Cactis algorithm is still completely outperformed by the CK algorithm. This is because, when using Cactis, clustering overhead increases very quickly with the number of objects, thus annihilating any gain achieved from good clustering capability. However, we have to keep in mind that this algorithm has been designed to run when the database is idle so that reclustering does not alter the database performance. Hence, if clustering overhead was not taken into account, the Cactis algorithm should perform about as well as CK

algorithm as long as the statistics used during the last reorganization are pertinent.

In terms of disk space, the more a clustering algorithm is simple, the less space it should use. Actually, the more a clustering algorithm is complex, the more it clusters objects according to sharp criteria. Thus, a smaller number of objects are likely to be clustered in the same clustering unit (either a page or a segment). So the number of pages needed to store the database is greater. Our simulation experiments confirm that fact. The ORION algorithm is the less greedy algorithm in terms of disk pages used. Then the Cactis algorithm follows, using almost half the number of disk pages needed by CK to cluster the database. However, when reorganizing the database, the Cactis and ORION algorithms need to build a new set of pages before deleting the old one. Thus they require about twice as much space as our graphs show.

Future research about this subject could be in a first step completing the simulation tests, notably by determining the effects of database changes, e.g., varying the probability for a class to belong to a composite class hierarchy, the average number and size of attributes, etc.

Then the next step would be the design and evaluation of one or several new clustering algorithms. One alternative would be to build a dynamic clustering algorithm that would use the same statistics as Cactis (i.e., objects access frequencies and relationships use frequencies) to cluster objects together, but could be able to use them at run-time (e.g., an algorithm activated at object creation time, like CK). Such an algorithm could have Cactis' good clustering capability without being handicapped by an important clustering overhead.

Another alternative would be to modify the CK algorithm so that it does not use users' hints for inter-objects access frequencies any more and rely only on statistics, in order to make the algorithm even more accurate and performant. The problem with users' hints is that their accuracy depends the user (either the database administrator or a programmer) knowledge of the database. On the other hand, automatically gathered statistics show an exact image of the database status. Thus, by computing inter-objects access frequencies for each structural relationship and each object at run-time, a better performance should be achieved.

REFERENCES

[ANDE90], T.L. Anderson, A.J. Berre, M. Mallison, H.H. Porter III, B. Scheider, "The HyperModel Benchmark", International Conference on Extending Database Technology, Venice, Italy, March 1990, pp. 317-331

[ANDR91a], T. Andrews, "Programming with Vbase", In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, pp. 130-177

[ANDR91b], T. Andrews, C. Harris, K. Sinkel, "ONTOS: A Persistent Database for C++", In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, pp. 387-406

[BANE87], J. Banerjee, H.-T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, H.-J. Kim, "Data Model Issues for Object-Oriented Applications", ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 3-26

[BATO85], D.S. Batory, W. Kim, "Modeling Concepts for VLSI CAD Objects", ACM Transactions on Database Systems, Vol. 10, No. 3, September 1985, pp. 322-346

[BENZ90], V. Benzaken, C. Delobel, "Enhancing Performance in a Persistent Object Store: Clustering Strategies in O_2", 4th International Workshop on Persistent Object Systems, September 1990, pp. 403-412

[BERR91], A.J. Berre, T.L. Anderson, "The HyperModel Benchmark for Evaluating Object-Oriented Databases", In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, pp. 75-91

[CATT91], R.G.G. Cattell, "Object Data Management: Object-Oriented and Extended Relational Database Systems", Addison-Wesley Publishing Company, 1991

[CHAB93], S. Chabridon, J.-C. Liao, Y. Ma, L. Gruenwald, "Clustering Techniques for Object-Oriented Database Systems", 38th IEEE Computer Society International Conference, February 1993, San Francisco, pp. 232-242

[CHAN89a], E.E. Chang, "Effective Clustering and Buffering in an Object-Oriented DBMS",

University of California, Berkeley, Computer Science Division (EECS), Technical Report No. UCB/CSD 89/515, June 1989

[CHAN89b], E.E. Chang, R.H. Katz, "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS", ACM SIGMOD International Conference on Management of Data, Portland, Oregon, June 1989, pp. 348-357

[CHAN90], E.E. Chang, R.H. Katz, "Inheritance in computer-aided design databases: semantics and implementation issues", CAD, Vol. 22, No. 8, October 1990, pp. 489-499

[CHEN91], J.R. Cheng, A.R. Hurson, "Effective clustering of complex objects in object-oriented databases", ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, pp. 22-31

[DEUX90], O. DEUX et al., "The Story of O_2", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 91-108

[FORD88], S. Ford, J. Joseph, D.E. Langworthy, D.F. Lively, G. Pathak, E.R. Perez, R.W. Peterson, D.M. Sparacin, S.M. Thatte, D.L. Wells, S. Agarwala, "ZEITGEIST: Database Support for Object-Oriented Programming", 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, pp. 23-42

[GRUE91], L. Gruenwald, M.H. Eich, "MMDB Reload Algorithms", ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, pp. 397-405

[HE93], M. He, A.R. Hurson, L.L. Miller, D. Sheth, "An Efficient Storage Protocol for Distributed Object-Oriented Databases", IEEE Parallel & Distributed Processing, 1993, pp. 606-610

[HUDS89], S.E. Hudson, R. King, "Cactis: A Self-Adaptive Concurrent Implementation of an Object-Oriented Database Management System", ACM Transactions on Database Systems, Vol. 14, No. 3, September 1989, pp. 291-321

[HURS93], A.R. Hurson, S.H. Pakzad, J.-b. Cheng, "Object-Oriented Database Management Systems: Evolution and Performance Issues", IEEE Computer, February 1993, pp. 48-60

[KATZ91], R.H. Katz, E.E. Chang, "Inheritance Issues in Computer-Aided Design Databases", In "On Object-Oriented Database Systems", Edited by K.R. Dittrich, U. Doyal and A.P. Buchman, Springer-Verlag Topics in Information Systems, 1991, pp. 45-52

[KIM90a], W. Kim, J.F. Garza, N. Ballou, D. Woelk, "Architecture of the ORION Next-Generation Database System", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 109-124

[KIM90b], W. Kim, "Object-Oriented Databases: Definition and Research Directions", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 3, September 1990, pp. 327-341

[MAIER86], D. Maier, J. Stein, A. Otis, A. Purdy, "Development of an Object-Oriented DBMS", ACM OOPSLA '86 Proceedings, September 1986, pp. 472-482

[SRIN91], V. Srinivasan, M.J. Carey, "Performance of B-Tree Concurrency Control Algorithms", ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, pp. 416-425

[TSAN92], M.M. Tsangaris, J.F. Naughton, "On the Performance of Object Clustering Techniques", ACM SIGMOD International Conference on Management of Data, San Diego, California, June 1992, pp. 144-153

[WILK88], W. Wilkes, "Instance Inheritance Mechanisms for Object Oriented Databases", 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, pp. 274-279