ICOT Technical Report: TR- 040

TR-040

Some Comments on Semantical Disk Cache Management for Knowledge Base Systems

by

H. Schweppe

(Technical University of Braunshweig)

January, 1984

©1984, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

TR-040

Some Comments on Semantical Disk Cache Management for Knowledge Base Systems

by
H. Schweppe
(Technical University of Braunshweig)

Some Comments on Semantical Disk Cache Management for Knowledge Base Systems

H. Schweppe
Technical University of Braunschweig

Abstract

The problem of accessing highly structured knowledge bases which reside on large capacity but slow storage devices, is addressed in this paper.

Knowledge representation techniques provide some means for defining clusters of semantically related objects. This issue is discussed in the paper, particularly how these relationships could be utilized for enhancing access to the knowledge base is discussed.

Multilevel storage as well as a single level storage architecture are discussed, both employing a cache attached to the peripheral storage devices.

In addition, an implementation of a fail-safe storage for transaction processing in this environment, is outlined. The paper intends as a basis for discussion rather than a well-defined solution for the I/O problem in knowledge base systems.

1. Introduction

A basic problem of data processing systems is how to achieve high performance interfaces to large volume data storage systems. This problem will become even more relevant in connection with large scale knowledge base systems. In order to reduce the number of accesses to slow storage devices, storage systems with large semiconductor buffer are now being used. An important factor decisive for the buffer hit ratio is the locality of references. In this statement, it is suggested, to improve the hit ratio by utilizing semantic criteria for buffer refreshment.

Representation of knowledge using higher level concepts such as object classes and instances provides some means for defining semantically meaningful clusters of objects. These clusters (or contexts) may be used as well as criteria for physically clustering data on disks as well as for improving buffer hit ratio. Two architectural models are discussed: multi-level storage and single level storage which makes access to secondary storage transparent for programs. An implementation of fail-safe storage for transaction processing based on shadows of objects, is outlined.

The paper is very sketchy, because it is intended as a basis for discussion not as a well-defined solution to the problems mentioned above.

2.Levels of Implementation and basic assumptions

Implementation of a complex structured system can be subdivided into a hierarchy of virtual machines. Each of these machines is defined by a set of data structures and operations. Mappings have to be defined between adjacent levels.

We subdivide a knowledge base system in a very simplified way into a two-level hierarchy:

- the knowledge representation level (KR)

- the knowledge storage level (KS)

The first level includes all structures and operations used to conceptually describe and manipulate knowledge. This includes facts and rules of inference as well as aggregations of objects, such as object instances, object classes, meta classes (see [SKAM 83]) or modules (see chap. 6 in [KLDG 83]) and operations, such as content inquiry.

The second level includes all mechanisms needed to map the higher level constructs to physical images, which may be stored in and retrieved from high volume storage devices. This level is essentially some kind of database machine.

We are not going to discuss the mapping of higher level objects to constructs executable on a machine (without reference to data on secondary storage devices). A convincing methodology has been suggested by Logic Programming.

Some basic assumptions on future technology have to be made in order to set up some frame for discussion.

Storage of large quantities of data (> 100 MB) requires devices which have considerably longer access time than fast memory. because of mechanical movement of the actuators involved, the ratio is 1:10**5. It is assumed furthermore that fast access storage will be volatile as opposed to slow large capacity storage devices.

The assumptions have a heavy impact on the overall performance of a system. Although the concept of performance does not consistently fit into a computational model, it is dangerous to ignore it during overall design of a system. This holds in particular for very slow operations such as external I/O.

3. Knowledge representation

We adopt the model of knowledge representation (KR) suggested in [SKAM 83] and [KLDG 83], at least our understanding of the representation model.

The following kinds of *objects* are defined:

object classes: they consist of facts and rules common to a set of objects (e.g. a general frame specifying the concept of a 'disease' in a medical consulting system).

object instances: concrete examples (instantiations) of a class (e.g. a specific disease such as pneumonia)

meta classes: abstractions of concepts used to define classes (e.g. the way of defining and using rules in a medical KB)

modules: the set of instances, classes and meta classes relevant to a particular domain.

Objects are linked by different kinds of mechanisms. There are explicit relationships defined between objects on different levels of abstraction (e.g. an instance *is-element* of a class) or implicit relationships are induced by identical values used as arguments of some predicate. In general, relationships may be associatively connected, i.e. X is related to Y if there are objects X1,...,Xn and relationships R1,...Rn+1 and (R1,X,X1), (R2,X1,X2), ..(Rn+1,Xn,Y).

No assumptions are made about how relationships of objects are implemented. The objects itself are naturally implemented by sets of facts and clauses in a logic programming environment.

Given an object, we may define its _context_. Roughly speaking, the context of an object X is the set of all objects Y, where there exists a relationship between X and Y.

Since there is a variety of ways to define specific relationships, this characterization of a context is too general. It may easily include the entire knowledge base.

A language is therefore needed which allows to define different shaped contexts of an object. We are not going to propose such kind of language for specifying semantical relationships between objects, but rather give some informal examples (upper case for variables, lower case for constants).

ctx (obj,C): the set of all facts given by the definition of class C of which obj is an element, furthermore those facts and rules derived by means of inference rules of C using the facts given by obj. (this is the simplest type of ctx which includes the inheritance of properties of objects from their class)

ctx(obj,r): the set of all objects, related to obj by the specific relation r

ctx(obj, B(r1,..,rn)): the set of those objects related to obj by a boolean condition of relationships

ctx(obj,r1,r2): the set of objects X related to obj by r1 joined with ctx(s,r2). (This is a simple kind of transitive context.)

The notion "context" obviously corresponds to some kind of _clustering_ of objects. This clustering may be syntactically defined using, for example, pattern matching criteria. However, the cxt operator (which has not been precisely defined in this paper!) is used to set up a cluster of *semantically related* objects to a given one.

From a naive point of view, semantic clusters are a natural way of organizing knowledge, although the primary reason of introducing them is efficiency in handling the KB. Time used for solving some application problem (e.g. suggesting a decision and its reason in a sophisticated consulting system) is one widely used measure of efficiency. Since time used will heavily depend on the number of accesses to an external KB, semantic clusters, if properly mapped to physical storage, may be of considerable importance.

Obviously it is unreasonable to expect one single mechanism which sets up an arbitrary context in uniform time. (Since there is only one way of *physical* clustering, clusters have to be implemented by access path rather than physical neighbourhood, which is the most efficient way on mechanically moving storage devices). The situation is analogous to human problem solving: while only a limited amount of facts are immediately accessible in one's head, there exists knowledge about how to obtain more information on a given subject. A 'quick access path' is the pile of books on the desk while it is usually more time-consuming to find a book in a library.

This analogy (which has been pointed out by K. Furukawa) gives considerable insight into the problems of knowledge management. A typical way of using a 'library knowledge base' is browsing. Only a couple of pages of thousands of books may contribute to the solution of a problem and these are *not* found by merely using some clever indexing scheme but rather by some kind of controlled search in a large set of data.

Response time for this type of operation will certainly differ from simple fact retrieval. This is in particular true, if interactive prompting of partial results by the user is the typical pattern of interaction with the KB during problem solving.

4. The Knowledge Storage System

There are two different ways of interfacing a data or knowledge processing system (KPS) and a knowledge storage system (KSS) from an architectural point of view. As far as the processing environment is concerned it is most convenient to have a one-level storage and thus not having to care explicitly about I/O-operations. We will discuss the single-level storage approach below.

A different, however more conventional approach is to provide an explicit interface to the background storage. A well-defined (logical) interface between a logic-programming-based inference system and a knowledge base (implemented as a relational database) has been developed in [MKMS 83], [KYKM 83].

4.1 Multi-level KSS

It is assumed that the overall system has three levels of storage:

- main memory which is directly accessible during a program execution

- disk cache memory which stores large volumes of data in fast memory

- large capacity background storage (disks)

Levels two and three are not directly accessible by the programs being executed. They are rather accessed using some logical I/O interface, but they are not aware of the two different levels of background storage. This kind of storage hierarchy is being implemented in the Relational Database Machine Delta ([KAKU 83]).

High-level I/O commands are executed in a specialized system tuned for executing relational algebra expressions. Execution includes access of background and cache storage as well as the preparation of result data (sorting, join of relations, restrictions).

In the previous section, we argued for having more complex objects for representing knowledge. Semantic clustering was suggested for relating objects to each other. Our underlying assumption is that an object X which is connected by some semantical relationship to Y, has a higher probability of being accessed, if Y has already been accessed. Semantic clusters are therefore significant for physical representation of knowledge.

There are two main aspects of physical representation which have an influence on KSS performance.

a) physical placement of semantic clusters on disk

Disk access time is determined to a large extent by latency, data transfer and positioning time (the largest fraction of overall access time). Physical neighbourhood of related data (avoiding positioning) will therefore support the access of the semantic clusters. Partial match hash functions may be employed for defining the physical mapping. Clustering is strongly related to _segmentation_ techniques for databases (see e.g. [TANA 83]). Attribute values of tuples are used in this case as a semantic criterion for data clustering.

This simple clustering scheme has to be extended for knowledge engineering applications. There has to be defined a *measure of semantical distance* between objects. One essential relationship is class membership. Object classes and instances should therefore be stored physically close together, since the inheritance of class properties to instances is a common operation.

There is, however, one important difficulty with physical clustering in a multiuser environment: independent requests may access objects in random order, thus nothing will be gained by physical neighbourhood of related data.

b) caching of semantically related objects

Until now we disregarded the disk cache. But the problems caused by multi-user access can be circumvented by making use of the cache properties.

We propose an *object-oriented cache*, i.e. data exchanged between slow disks and fast cache memory are *semantically related objects* rather than disk pages (or disk tracks as in some disk cache systems).

The interface between KPS and KSS may be extended by staging commands. The effect of such command is the staging of objects semantically related to a given object x, which is accessed. The scope of objects to be staged is either defined by some measure of semantical distance or explicitly e.g. by specifying the kind of relationships. The distance or semantical relationship between objects may be defined in the same way as in the definition of contexts on the level of knowledge representation.

From a performance point of view it is important that supplying an object to a requester and establishing a semantic context in the cache may be done completely independent. The callee will typically already resume processing, while the cache is updated. The intended effect of caching is the increase of hit probability in subsequent processing steps.

The cache hit ratio may be controlled partially by the executed program, if it uses application dependent knowledge of how data objects are accessed, and if contexts may be specified in staging commands.

There is, however, always a tradeoff between the performance gained by this kind of optimization and the impact on the clarity of an algorithm, even if staging commands can be integrated in the programming environment in a clean way.

The interface between the KPS and the KSS in a multi-level store can be summarized as follows:

- a relational query interface as described in [KYKM 83] which may be extended by constructs for higher order objects than relations if such objects are used for knowledge representation.

- data staging commands as: stage(<object>, <ctx-spec>, mode)

A 'mode'-parameter is useful to define the types of operations which are going to be applied to objects, in particular read or write access. This is very useful when processing transactions (see below).

A further extension of the interface between knowledge processing and storage systems has been suggested in [YOKO 83]. Since facts <u>and</u> rules are stored in the KB, I/O traffic will be decreased, if unification operations may be performed also in the KSS. Some clauses have then not to be transferred to the KPS, thus decreasing overhead. It seems to be a promising way to extend the KSS by processing facilities. We will not discuss this aspect in this paper.

4.2 One-level KSS

From the viewpoint of a programming language it is most convenient to address data in a uniform way. This way of addressing has been accomplished by using very large address spaces which make I/O transparent for the programs. But even in systems with very large (e.g. about 100 MB) real memory thrashing may easily become a problem, if locality of accesses can not be achieved in the huge virtual space. In knowledge base systems, thrashing is not primarily caused by sharing the real memory between several users, but rather because semantically related data will be usually spread across the address space. In contrast to accessing conventional program code, the typical sequence of accesses does _not correspond to physical sequentiality_. Therefore a virtual memory system using (physical) page replacement is not suited for knowledge engineering applications.

We therefore propose an _object-oriented_ paging mechanism. This mechanism which employs ordinary page transfer, uses semantic information in order to decrease the number of addressing faults.

A simple example is the access to a fact in a relation. If a fault occurs, the paging mechanism not only makes available the fact requested, but will in addition put the class description of that object instance and further instances into real memory.

As is easily seen, there is a strong similarity between the management of the multilevel and the one level storage system. In fact, both schemes make use of semantic relationships between objects in order to decrease random access of slow background storage. The main difference is the degree of control, a program has on staging (paging) of data. While in the second approach a general mechanism for finding semantically related objects is assumed, staging commands are explicitly used in multi-level storage.

This is, however, not an essential point, because the same mechanism employed in one-level store may be used in a multi-level environment without effecting the logical interface between KSS and KPS.

The primary question is, however, whether a general mechanism can be implemented, which finds a semantical cluster of a given object. Since the term 'semantically related' is vague, it is more reasonable to have explicit control on staging operations.

5. Transaction Processing

Until now we disregarded the difference between read and write access and the concept of transactions as atomic operations (e.g. either all actions of a transaction end successfully or the KB state is preserved).

Transaction processing in database systems has been discussed in depth in the literature (see [GRAY 81] for an overview). One of the primary problems is to establish a (logically) fail-safe storage system, that is, it will recover from failures to a consistent state (committed transactions are physically represented in the KB, all uncommitted transactions do not leave any dirty data).

We do not discuss concurrency among readers and writers of data, because adequate solutions have been proposed.

The method we propose for transaction commitment follows the lines of solutions presented by [LORI 77] and [BAEL 83]. It is based on the mechanism of shadow pages. Each time an object is staged to fast, volatile storage and a write mode is signaled, two copies of the object are generated. As opposed to the original shadow page technique, updates are made in place. That is, the modified objects will be stored in the same physical location as the original one. At commit time of a transaction, the updated object versions ('after images') are written into a _sequentially_ organized recovery area. As a consequence, saving objects before the commitment is made, is considerably faster than writing the objects into the KB (random access to physical locations on disk).

If the sequential write has been successfully completed, the before image (state of objects before start of transaction) are released in the fast memory (cache or one level store). The new version is not written back into the KB before the 'object replacement' algorithm (which may be simply LRU) decides to replace the object. In case of transaction backout the after images are merely released.

Obviously, a warmstart routine can easily restore cache (or main memory) from the recovery area.

In principle, the same technique can be applied when using the virtual storage approach. The situation is however complicated by the fact, that it is not known in advance, whether read or write access is made to an object. It is therefore required, to copy an object (or part of it), before the after image (i.e. the updated part of the object) is established. It is not obvious, whether this kind of implementation of a logically non-volatile storage is feasible, since the mechanism has to be built into the addressing subsystem of the memory which has to implement address translation and semantically induced staging of objects in the approach described.

6. Conclusion

The main area of research which is crucial for narrowing the gap between KPS and KSS (or KBM) is the investigation of semantical clustering of knowledge. When the semantical relationship between objects will be precisely defined, mapping mechanisms of objects to secondary storage have to be designed based on these semantical relationships. Staging of objects according to these relationships will determine the performance of large scale knowledge processing system, since I/O will have at least as much impact on performance as internal processing.

References

[BAEL 83] Bayer,R.; Elhard,K.: A Database Cache for high Performance and fast restart in database Systems, (to appear)

[GRAY 81] Gray,J.N.:The transaction concept: Virtues and Limitations, Proc. VLDB, 1981

[LORI 77] Lorie,R.A: Physical Integrity in a large segmented Database, ACM Trans. on DBS, 2(1), 1977

[KAKU 83] Kakuta,T.; Miyazaki,N.; Shibayama,S.; Yokota,H.; Murakami,K.: A Relational Database Machine "Delta", ICOT TM-008, 1983

[KLDG 83] Kernel Language Design Group: Conceptual Specification of the 5th Gen. Kernel Language, Vers. 1 (KL1), ICOT Technical Report (to appear), [KYKM 83] Kunifuji,S.; Yokota,H.: PROLOG and Relational Databases for 5th Generation Computer Systems, Proc. of Workshop on Logical Bases for Data Bases ,Toulouse, Dec. 1982 (revised version to appear as ICOT Technical Report)

[MKMS 83] Murakami,K. et al.: A Relational Database Machine: First Step towards a Knowledge Base Machine, Proc. of the 10th International Symposium on Computer Architecture, Stockholm, June 1983

[SKAM 83] Sugiyama,K. et al.: A Knowledge Representation System in Prolog, ICOT TR-0024, 1983

[TANA 83] Tanaka,Y.: Adaptive Segmentation Schemes for large Relational Database Machines, in: Database Machines, Proc. IWDM 83, Springer Verlag

[YOKO 83] Yokota,H. et al.: An Investigation for Building Knowledge Base Machines, Proc. of the 27th National Conference of Information Processing of Japan, Nagoya, Oct. 1983 (in Japanese)