Synergistic Buffer Management for Data Access in Object-Oriented Database Systems

Introduction:

In the modern world the need of organized data is increasing rapidly but to retrieve the information from huge data source is still a big problem. Today we have large number of data source (Database), but how can we minimize the time in accessing the information from the complex database. Unfortunately: There is no silver bullet Due to the daily advance in the computing power of the hardware it become more complex to find the solution of the problem. Despite the dominance of relational database management systems (RDBMS) in the database market, object-oriented database management systems (OODBMS) continue to play an important role in complex data management.

An object database (also object-oriented database management system) is a database management system in which information is represented in the form of objects as used in object-oriented programming.

When database capabilities are combined with object-oriented programming language capabilities, the result is an object-oriented database management system (OODBMS). OODBMS allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Because the database is integrated with the programming language, the programmer can maintain consistency within one environment, in that both the OODBMS and the programming language will use the same model of representation.

Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application.

As the usage of web-based technology increases with the implementation of Intranets and extranets, companies have a vested interest in OODBMS to display their complex data. Using a DBMS that has been specifically designed to store data as objects gives an advantage to those companies that are geared towards multimedia presentation or organizations that utilize computer-aided design.
The ever increasing demand for fast complex data storage and retrieval makes a strong case for OODBMSs' (Object Oriented Database Management Systems) survival as an important database management technology. OODBMSs are particularly suited to the management of complex data since they provide fast navigational access, efficient storage of class methods and efficient and natural storage of many-to-many relationships.

Storage management is an important issue in the design of any object-oriented database management system (OODBMS). In fact, most object-oriented database management systems are composed of two main subsystems, an interpreter and a storage manager.

The interpreter provides the operational semantics as seen by the user; it understands the details of the data model, enforces object encapsulation, and executes methods. It calls the storage manager for physical data access and manipulation.

The storage manager, in turn, concerns itself with the placement of objects on secondary storage, movement of data between secondary storage and main memory, creation of new objects, recovery, concurrency control, and sometimes indexing and authorization.

Complex data is most often accessed via navigation. Relational Data Base Management Systems (RDBMSs) are poorly suited for fast navigational access since simple object graph navigations can often turn into joins of multiple tables when converted to queries on the relational schema.

Object-Relational Database Management Systems (ORDBMS) offer better navigational performance by storing references between objects inside the relational tables. Object navigations can then proceed by de-referencing these references instead of executing multiple joins. However this approach still does not perform as well as OODBMSs which often store objects traversed together on the same disk page, thus generating less IO. In contrast, ORDBMSs typically do not store objects traversed together on the same disk page, they are often stored as tuples on different relational tables instead. Typically, tuples of the same table are stored together on disk. Thus one of the most attractive characteristics of OODBMSs is fast navigational access.
The ability for OODBMS to provide fast navigational access is conditioned on efficient main memory caching, which is made more important by the fact that disk IO performance improves at only 5-8% per year whereas CPU performance doubles approximately every 18 months. An example of a fast disk is the Seagate Cheetah 18 with an average access time of 8.19 ms. A consequence is that disk IO is likely to be a bottleneck in an increasing number of OODB applications. Thus the focus of the proposed study is on reducing the effects of disk IO on the performance of OODBMSs.

It should also be noted that much recent research on performance optimization of RDBMSs has been focused on the main memory bottleneck instead of the disk IO bottleneck [Ailamaki et al. 1999; Chen et al. 2001; Rao and Ross 1999; Rao and Ross 2000]. This is due to main memory becoming cheaper and sophisticated techniques for hiding disk IO latency in RDBMSs. Some database users now choose to set up their system so that the entire database fits in memory. However, disk continues to remain cheaper than memory, and so in any cost/performance analysis, scalability will ultimately dictate the use of disk. In addition, in many database applications a very high percentage of data accesses are directed at a very small portion of the database. In such cases, it is more cost-effective to only store a small portion of the database in memory (the portion which has a very high percentage of data access).

When the database is larger than memory, techniques for hiding disk IO are needed to ensure the system is not bottlenecked at the disk. Existing techniques for hiding disk IO in OODBMSs do not perform as well as their RDBMS counterparts. This is because navigational data accesses (often used in OODBMSs) are much harder to predict than index and table accesses in RDBMSs. The disk IO bottleneck in OODBMSs is thus a pressing research problem.

Effective buffer management is the key to reducing the disk IO bottleneck in OODBMSs. There has been much existing work, namely in the areas of: static clustering; dynamic clustering; buffer replacement; and prefetching. All of these techniques can be used together in a complimentary
manner. Most existing research has focused on finding the best solution for each area with little regard on how solutions from the different areas affect each other. We believe synergy exists between the areas, and that exploiting the synergy leads to the best overall solution. We focus on demonstrating synergistic techniques are both feasible to implement and outperform their non-synergistic counterparts.

**Buffer Management Techniques**

There are four proven techniques of improving the IO performance of OODBMSs: static clustering; dynamic clustering; pre-fetching and buffer replacement.

Objects are uniquely identified and accessed by object identifiers (oids). The (persistent) identifiers are “physical” identifiers, that is, reflecting the location on disk. Such identifiers may have a performance edge over “logical” identifiers, in the sense that they may save an extra disk access for retrieving the “object table” (which maps the logical identifiers to the physical addresses). This disk access is a necessity in the case of logical identifiers. A major problem, though, is moving objects on disks without changing their identifiers.

Clustering is the arrangement of objects into pages so that objects accessed close to each other temporally are placed into the same page. When a requested object is loaded from disk, other objects (on the same page) which are likely to be needed in the near future are also loaded. This in turn reduces the total IO generated. Clustering algorithms can be separated into two types, static and dynamic. In static clustering, re-organisation takes place when the database is offline. In contrast, dynamic clustering re-organises the database while it is online. Pre-fetching involves predicting the user's next disk page request and then loading that page into memory in the background. In this manner disk IO can be overlapped with CPU, thus reducing disk IO stall time. Buffer replacement involves the selection of a page to be evicted when the buffer is full. The page evicted ideally should be the page needed furthest in to the future. Selection of the correct page for eviction results in a reduction in the total IO generated by the system.