Prefetching Techniques for Object-Oriented Databases

Nils Knafla

Abstract: The memory hierarchy means that there is a trade-off between cost and speed for different memory types. Prefetching is the generic term used to describe techniques that use fast, expensive memory more effectively by ensuring that the process of retrieving relevant blocks from slower memory is initiated sufficiently early to avoid delays due to misses in the faster memory. Prefetching has become even more important with the advent of client/server computing and object-oriented databases. Pages of objects must be retrieved from secondary storage on the server and placed into the virtual memory address space of the client application.

Introduction

The technology advances of semi-conductor memory and magnetic disks have a big impact on the performance of Object-Oriented Databases (OOD). In Fig. 1 we see a two-order gap in access time between memory and disks. Memory is faster but disks offer cheaper storage for pages.

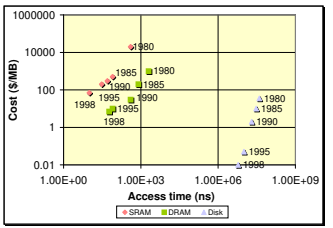

Figure 1. Performance/price development

OODs store the data and references of a persistent object on secondary storage. A persistent object has an identity that pre-dates and outlasts the execution of a client application. Most OODs cluster objects into units of pages on disk. The page is also the unit of network transfer between a database client and server. Fig. 2 shows the basic components of a client/server OOD. If the application wants to access a persistent object it first checks its local object buffer and page buffer. If the page is not resident it sends a request to the server to retrieve the page. This page fetch operation can seriously delay the application but the delay can be avoided by an efficient prefetching technique.

The challenge in this PhD research lies in the fact that with OODs, the access pattern of an application is usually hard to predict accurately and we must avoid wasteful prefetches to conserve space in the client page buffer. The application navigates through the object graph by dereferencing pointers to other objects. From the current position of the client navigation we compute the access probability of all adjacent pages. If the access probability of a page is higher than a threshold then the page is a candidate for prefetching.

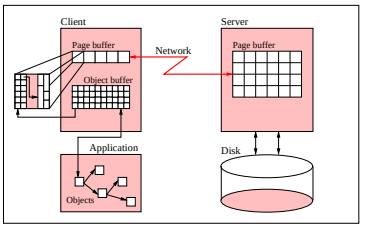

Figure 2. Client/Server OOD Architecture

Prefetching Algorithm

We use a discrete-time Markov chain to model objects and object relationships. Several closely-integrated components characterise our prefetching technique (a more detailed description can be found in [1]):

- The probability and mean time to access a page is computed by a method called *hitting times*. This computation is based on the transition probabilities of the object relationships.
- We compare the page access probability with a prefetch threshold which is determined by the benefit of a correct prefetch and the extra cost of an incorrect prefetch.
- We also consider the timing of a prefetch. Starting the request too late would reduce the amount of savings and too early could result in an incorrect prefetch.

Multithreaded ESM Architecture

We have extended the well-known EXODUS object storage manager (ESM) from the University of Wisconsin in the following ways:

- We have incorporated the above prefetching technique.
- The impact on the performance of the client machine is minimised by exploiting multithreading and shared-memory parallelism. We use Solaris threads for prediction, prefetching and flushing pages.
- The database client and server transfer pages via the *multiple* sockets.

Performance Analysis

A comprehensive description of all ESM implementation results can be found in [2, 3, 4]. The general result was that prefetching can reduce the cost of a page fetch by up to 60% if the page is resident on access. The synchronisation costs of the threads are responsible for not achieving a total reduction.

We also verified our algorithm by a simulation (Fig. 3). We created a complex object structure and varied the transition probabilities of references. The prefetch applications always outperform the Demand application (without any prefetching). The applications with + sign also consider the best time to start the prefetch. The prefetch threshold parameter ensures that the prefetching applications will never increase elapsed time but can reduce it drastically when object access is reasonably predictable.

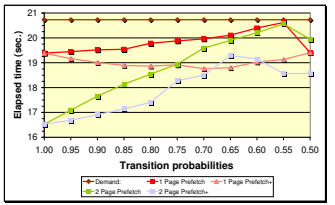

Figure 3. Simulation result

References

[1] N. Knafla. Predicting Future Page Access by Analysing Object Relationships. Technical Report ECS-CSG-35-97, Dec. 1997.

[2] N. Knafla. A Prefetching Technique for Object-Oriented Databases. *15th British National Conf. on Databases*, London, UK, July 1997.

[3] N. Knafla. Speed Up Your Database Client with Adaptable Multithreaded Prefetching. In *Proc. of the Sixth IEEE Int. Symp. on High Performance Distributed Computing*, Portland, Oregon, Aug. 1997.

[4] N. Knafla. An Adaptable Multithreaded Prefetching Technique for Client-Server Object Bases. *Cluster Computing*, 1(1), 1998.