Object Groups May Be Better Than Pages

Mark Day
mday@lcs.mit.edu
MIT Laboratory for Computer Science[*]

July 30, 1993

Abstract

I argue against trying to solve the problem of clustering objects into disk pages. Instead, I propose that objects be fetched in groups that may be specific to an application or user, and that can be computed at fetch time. I briefly describe *crystals*, which serve to record such groups statically. Finally, I speculate that object fetching may be particularly relevant for providing services to very small machines with limited memory, such as personal digital assistants.

1 Context

In a distributed object-oriented database system, server machines store persistent objects shared by applications running on client machines. When an application invokes an operation on a persistent object, that operation must run on either the server or client. I consider the case where objects are moved or copied to the client for at least the duration of the client transaction. A number of existing object-oriented databases work partially or entirely in this mode of executing operations on the client machine: examples are O2 [1], GemStone [2, 8], and Orion [6].

[*]545 Technology Square, Cambridge, MA 02139, USA. This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-91-J-4136 and in part by the National Science Foundation under Grant CCR-8822158.

A number of systems transfer pages from server to client. Transferring pages simplifies storage management, since the units transferred are of fixed size, and it allows the use of hardware page-fault mechanisms. However, this approach requires that one have tremendous faith in mechanisms for *clustering* objects into pages in a reasonable way.

2 Sophisticated Clustering is a Zero-Sum Game

Assume for the moment that we have perfect predictive information about access patterns of applications. There is a certain amount of clustering that we can do that benefits all of the applications, but after this point clustering becomes a zero-sum game: making the clustering better for one application makes it worse for another. This effect is not just an intuition, but was apparent in recent work by Tsangaris and Naughton [12].

If it were possible, we would like to cluster as well as we can for each application. In particular, we should be able to accommodate a new application or a dramatically changed access pattern in a relatively straightforward and inexpensive way.

3 Does Clustering Really Work?

The problem is that we never do have perfect predictive information. Among those who believe that clustering is important, the field then divides into two camps: one consists of those who assert that people are unlikely to cluster a database well, so database clustering should be done automatically; the other consists of those who assert that automatic mechanisms are likely to be too expensive, and so clustering should be done by a human database administrator.

As far as I can tell, both of these camps are partly right. I have yet to see any indication that automatic tools are practical in production systems, nor any indication that clustering by an administrator is something other than a shadowy art that sometimes does the right thing and sometimes does not.

Clustering is intrinsically a difficult, and perhaps impossible, problem. The notion is that we should somehow be able to predict access patterns to a very large collection of objects, and then pack the objects into pages accordingly. Why on earth do we want to do that, anyway?

4 Are We Solving the Wrong Problem?

The first thing to realize about clustering is that it is a compromise. It is *not what we really want to do*. Instead, it is the best that we can do with disks.

A disk is an example of a system where the initial cost of a transfer is relatively high and the marginal cost of transferring extra (sequentially-accessed) information is relatively low. This cost structure provides an incentive to make each disk read fairly large, to amortize that cost over a number of contiguous objects.

We may have some way to determine "likely" groups of objects to prefetch based on the application. But because there's no way with a disk to delay the computation of object groups until runtime, someone has to make their best guess about access patterns, and cast those in oxide on the disk. Since disks are typically implemented to manage fixed-size blocks or pages, the clustering takes place in terms of those blocks or pages.

But the object groups that we really want to fetch vary among applications, and even vary from day to day. There is no "right" clustering that we can fix in place and forget.

5 Solving the Right Problem

One way to view a distributed system is that it just moves the disk further away from the application; but that view ignores the possibility of taking advantage of the server's memory and processor. Consider instead how object fetching can change to take advantage of that memory and processing power. Assume for the moment that the server can keep all of the objects in the current working sets of all its clients in real memory.

A network has some similarities to a disk: there is a relatively high initial cost for sending any packet and a relatively low marginal cost for sending more bits in that packet. The economic analogy is only true up to the point where the first packet is full, since subsequent packets are no cheaper — unlike reads of subsequent blocks as a disk rotates under the head. But in contrast to the disk, the server's memory is random access. In addition, the server processor can be used to determine which objects to fetch. So we can build a system in which the server dynamically determines groups of objects to prefetch, instead of simply sending the page on which the requested object appears.

Sending something other than a page is becoming more important as hardware pages get larger: the newest machines in our lab have pages of 8K bytes (1K 8-byte words), and we expect that about 100 "typical" objects will fit on each such page. This is in contrast to when Stamos made measurements of LOOM and estimated that a typical page would hold 11 objects [10]. As pages get larger, it seems less likely that a single page will contain only semantically-related objects, and more likely that a single page will contain a lot of junk from the point of view of any given application. Taken together with the possibility of poor clustering spreading the objects of interest across multiple pages, big pages can lead to very inefficient use of the network and the client's local storage.

In the Thor distributed object-oriented database system[7], we have one mechanism for moving data between the server's memory and disk, and a different mechanism for moving data between the client and the server. Server *segments* are significantly larger than current pages. Segments are transferred between the disk and the server's memory. Objects are then fetched out of these large segments and sent to clients in smaller groups that are more useful for those clients than whole segments would be.

6 Crystals

Although it is possible to delay all computation of group membership until the moment of fetching an object, it will often be useful to record the result of that computation and use it instead of rerunning the computation. I propose a mechanism, called *crystals*, to accomplish this. A crystal is an object that describes a group of objects specifically for the purpose of prefetching those objects. It can be constructed explicitly by identifying a collection of objects, or it can be constructed implicitly (all of the objects touched between starting and ending an implicit crystal are included in the crystal). It may also be possible to construct and use a crystal completely automatically, in a way analogous to the working graphs and program trees of Tait and Duchamp [11]. However, as currently envisioned, crystals are user- or application-level hints, and so have more in common with the ideas of transparent informed prefetching [9].

One way to see a crystal is as a substitute for a cluster: we compute a placement and save it. A crystal is better than a cluster because multiple independent users can have conflicting crystals, but the system can have only one physical clustering. We thus provide a tool to applications to describe their own "clustering" and use the result to prefetch accordingly. However, in contrast to previous work (such as segments and segment groups in ObServer [3]) we do not guarantee that the elements grouped by a crystal will be contiguous on disk. We use the crystal simply as a hint: we prefetch the objects present in memory, and we can start disk reads for the objects not in memory.

A secondary use of crystals is to provide extra information about likely access patterns to the process that clusters objects into segments on the disk.

7 Implications for Small Client Machines

The next generation of very small machines — pen-based machines, personal digital assistants, wearable computers, and the like — seem likely to repeat the path already followed by minicomputers and personal computers: they will initially have relatively little physical memory, and it will be years before they have virtual memory. They may often be communicating over links that are quite slow compared to the speeds available for desktop machines.

Such small machines can benefit from being clients of a system like Thor, where the servers can provide large amounts of highly-available, reliable, geographically-distributed shared storage. Thor can accommodate even machines with very limited storage and no interest in prefetching, since it can send individual objects in response to requests. But by using prefetching and crystals, clients with more resources can also achieve higher performance than would be possible in a strict object-fetching system.

OOZE [4] and LOOM [5, 10] already demonstrated the value of object faulting for small memories. Tunable prefetching seems likely to be useful for a wider range of clients than clustering into fixed-size pages could be.

References

[1] François Bancilhon, Claude Delobel, and Paris Kanellakis, editors. *Building an Object-Oriented Database: The Story of* O_2. Morgan Kaufmann, 1992.

[2] Paul Butterworth, Allen Otis, and Jacob Stein. The GemStone object database management system. *Communications of the ACM*, 34(10):64–77, October 1991.

[3] Mark F. Hornick and Stanley B. Zdonik. A shared, segmented memory system for an object-oriented database. *ACM Transactions on Office Information Systems*, 5(1):70–95, January 1987.

[4] Ted Kaehler. Virtual memory for an object-oriented language. *Byte*, pages 378–387, August 1981.

[5] Ted Kaehler. Virtual memory on a narrow machine for an object-oriented language. In *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 87–106, 1986.

[6] Won Kim, Nat Ballou, Hong-Tai Chou, Jorge F. Garza, Darrell Woelk, and Jay Banerjee. Integrating an object-oriented programming system with a database system. In *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 142–152, 1988.

[7] Barbara Liskov, Mark Day, and Liuba Shrira. Distributed object management in Thor. In M. Tamer Özsu, Umesh Dayal, and Patrick Valduriez, editors, *Distributed Object Management*. Morgan Kaufmann, San Mateo, California, 1993.

[8] David Maier, Jacob Stein, Allen Otis, and Alan Purdy. Development of an object-oriented DBMS. In *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 472–482, 1986.

[9] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *ACM Operating Systems Review*, 27(2):21–34, April 1993.

[10] James William Stamos. A large object-oriented virtual memory: Grouping strategies, measurements, and performance. Technical Report SCG-82-2, Xerox PARC, May 1982.

[11] Carl D. Tait and Dan Duchamp. Detection and exploitation of file working sets. In *Proceedings of the 11th Conference on Distributed Computing Systems*, pages 2–9, Arlington, Texas, 1991.

[12] Manolis M. Tsangaris and Jeffrey F. Naughton. On the performance of object clustering techniques. In Michael Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 144–153. ACM Press, 1992.