Disk Management for Object-Oriented Databases (Student Paper)

Sanjay Ghemawat *
(sanjay@lcs.mit.edu)
MIT Laboratory for Computer Science

March 1, 1994

An object-oriented database provides persistent storage for a large number of objects. These objects may be very small, and the access patterns are likely to be not as uniform as the mostly sequential reads and writes seen in file-systems [1]. For example, the OO7 benchmark for object-oriented databases specifies a number of traversals that follow pointers around a graph of objects [2]. Given these differences between file-systems and object-oriented databases, disk management techniques used in file-systems will not perform well if naively applied to object-oriented databases.

In this paper I propose three disk management strategies for object-oriented databases. These strategies are based on earlier work on file-systems. They differ from this earlier work in their support for a large number of small objects and non-sequential access patterns.

1 Background

Object-oriented databases are usually built around a transaction system because most meaningful database operations require reading and writing multiple objects. The proposed strategies exploit the feature of transactional systems where modifications do not have to be made persistent until transaction-commit. Techniques similar to the ones proposed here could be used in a system without transactions if modifications do not have to made persistent at once, but can be buffered in memory for short periods of time. (The thirty second write delay in some file-systems is an example.)

The rest of this section contains a brief description of techniques used in various disk-based systems that have been adapted to fit into the proposed disk management strategies.

1.1 Caching

Many systems cache data in volatile memory to reduce the number of disk accesses required to satisfy read requests. Caching can vastly improve read performance. However, for reliability reasons, modifications have to be made persistent at transaction commit and therefore caching does not directly affect write performance.

1.2 Clustering and Prefetching

Disks tend to perform rather poorly if the unit of transfer between disk and memory is small. Much higher disk throughput can be achieved by reading and writing large amounts of data in one disk operation. Therefore if certain objects are likely to be read together, these objects can be stored contiguously on disk and can be read into the cache in one disk operation. This technique can significantly improve the performance of a disk-bound system, but will result in wasted work if reference patterns change so as to not match the layout of objects on disk.

1.3 Write-Ahead Logging

Transactional systems can append all pending modifications to a *write-ahead log* at transaction commit. If the log is stored contiguously

*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.

on disk, all pending modifications can be combined into one (or at most a few) disk write. This technique reduces commit latency. However, since the updates in the write-ahead log must be moved to their final location on disk eventually, disk throughput can only be improved in combination with other techniques.

1.4 Replication

In some systems [7] the write-ahead log is replicated and stored in processor memories instead of being stored on disk, and therefore commit latency is improved even further. Again, replication only reduces commit latencies. Other techniques are needed to improve disk throughput.

1.5 Write Sorting

The disk scheduler can reorder writes to minimize rotational delays and disk arm movements. The scheduler can also discard duplicate writes and combine smaller contiguous writes into one large disk write and therefore obtain a higher disk transfer rate. This technique works especially well with write-ahead logging systems because in such systems transactions are committed by making the write-ahead log persistent. Therefore the write-sorting step that moves data from the log to the database proper can be delayed without increasing transaction commit latency. (Write-sorting becomes more effective as disk writes are delayed for larger amounts of time because at any given moment, more pending writes will be available for sorting.)

1.6 Log-Structured Storage

Some file-systems carry the write-ahead log scheme a step further [8, 9]. Persistent storage for the entire database is arranged as a log. At transaction commit, modified data is appended to the log. There is no copying of data from the log to a separate organized structure on disk. Data is reorganized only to create empty space as the log grows to fill the disk.

2 Proposed Strategies

The proposed strategies achieve different trade-offs between read and write performance. All of these strategies are based on the assumption that the set of persistent objects is partitioned into disjoint *segments*. Another assumption is that a partitioning process will cluster related objects into the same segment. (The details of the partitioning process are not described here — it may be based on application level hints, or the reference patterns seen by the database.)

Each of the proposed strategies uses a volatile cache to satisfy read requests quickly. Transactions are committed by storing updates into a replicated write-ahead log.

2.1 In-Place

The *in-place* strategy assigns a fixed disk location to each segment. Entire segments are read into the volatile cache at one shot under the assumption that if one part of the segment is being read, then other parts will be read soon. This strategy is derived from the usual *read-optimized* disk management strategies for file-systems and traditional databases [9].

This strategy should provide good read performance, and replication, write-ahead logging, and write-sorting will be used to improve the usually poor write performance of such a read-optimized strategy. Some complications may arise when segments change in size and have to be moved around on disk, or when access patterns change and objects have to be re-partitioned into new segments to provide effective clustering.

2.2 Log-Structured

The *log-structured* strategy organizes all available disk space as a log. Conceptually, the disk log is an extension of the replicated in-memory write-ahead log. As the memory log fills up, its contents are appended to the disk log. This strategy is a simple adaptation of the disk management strategy used in the log-structured file system [9].

This strategy has the potential to perform very well on writes because the contents of the memory log can be moved to the disk log with

big sequential writes. The drawback of this strategy is that the contents of a segment can end up scattered over the disk and therefore read performance may suffer. In addition, a recent file-system study has shown that under certain access patterns the overhead of reclaiming disk space in such a strategy can be high enough to completely negate any performance benefits provided by a log-structured disk organization [10]. It will be interesting to see if similar results are obtained for the access patterns seen by an object-oriented database.

2.3 Hybrid

The *hybrid* strategy is also influenced by work on the log-structured file system and views disk space as a log. However, the hybrid strategy always appends whole segments at a time to the disk log. Therefore the contents of a segment are always stored contiguously on disk.

The hybrid strategy will provide good read performance like the in-place strategy. The presence of large contiguous segments on disk may also allow us to use a more efficient space reclamation policy than the pure log-structured strategy. However, write performance will suffer compared to the pure log-structured strategy because of extra data movement required to keep segment contents stored contiguously on disk. Write-sorting and write-ahead logging may help alleviate this problem.

3 Conclusions

The proposed strategies explore the trade-off between the read and write performance of an object-oriented database. My plan is to implement the three strategies as part of the Thor object-oriented database [6] and evaluate their performance under the OO7 benchmark [2]. The traversals specified in the OO7 benchmark may not cover all access patterns of interest and therefore I may also have to invent some benchmarks of my own to evaluate these strategies.

My expectation is that the the log-structured strategy will not perform very well. It has performed well in file-systems because of the mostly sequential access patterns seen in those systems. In the presence of less sequential access patterns, the overhead of space reclamation and the poor clustering of related objects on disk may very well outweigh the good write performance provided by a log-structured strategy. It is harder to choose between the hybrid strategy and the in-place strategy. The hybrid strategy may be able to pick a better location for the segment it is writing out to disk. (Some of the techniques described in [3, 4, 5] may be applicable.) However, the space reclamation costs for the hybrid strategy may tip the balance in favor of the in-place strategy.

References

[1] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212. Association for Computing Machinery SIGOPS, October 1991.

[2] Michael J. Carey, David J. DeWitt, and Jeffrey F. Naughton. The OO7 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 12–21, Washington, DC, May 1993.

[3] Chia Chao, Robert English, David Jacobson, Alexander Stepanov, and John Wilkes. Mime: A high performance storage device with strong recovery guarantees. Technical Report HPL-92-44, Hewlett Packard, Software and Systems Laboratory, March 1992.

[4] Robert English and Alexander Stepanov. Loge: A self-organizing storage device. In *Winter Usenix Technical Conference*, pages 237–251, San Francisco, January 1992. USENIX Association.

[5] Robert B. Hagmann. Low latency logging. Technical Report CSL-91-1, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, February 1991.

[6] 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.

[7] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the Harp file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 226–38. Association for Computing Machinery SIGOPS, October 1991.

[8] John Ousterhout and Fred Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *ACM Operating Systems Review*, 23(1):11–28, January 1989. Also appears as University of California, Berkeley, Technical Report UCB/CSD 88/467.

[9] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 1–15. Association for Computing Machinery SIGOPS, October 1991.

[10] Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, and Carl Staelin. An implementation of a log-structured file system for UNIX. In *Winter Usenix Technical Conference*, pages 201–220, San Diego, CA, January 1993. USENIX Association.