Disk Management for Object-Oriented Databases

Sanjay Ghemawat *
Email: sanjay@lcs.mit.edu
MIT Laboratory for Computer Science

Object-oriented databases provides persistent storage for a large number of objects. This paper proposes efficient disk management techniques for such databases.

1 Introduction

Object-oriented databases differ from traditional file-systems in a number of important ways. Persistent objects are potentially quite small, say on the order of a 20–200 bytes; files are typically much larger. Object-oriented databases are accessed via transactions that are not considered committed until all relevant modifications are stored on stable storage. Such transactions may access a large number of objects. File-systems on the other hand are typically accessed via operations that each modify a very small number of files (usually just one). The rest of this section describes some well-known implementation techniques that are being incorporated into the Thor object-oriented database [4] to improve the performance of the proposed disk management strategies.

1.1 Caching and Prefetching

Objects are cached in volatile memory to reduce the number of disk accesses required to satisfy read requests. Furthermore, if the system can deduce the objects that are likely to be read soon by a client, then these objects are *pre-fetched* into the cache before the client actually asks for them.

Caching and prefetching can vastly improve read performance. However, modifications have to be made persistent at transaction commit for reliability reasons and therefore caching does not directly affect write performance. In addition, a poor prefetching strategy can create extra disk traffic.

1.2 Clustering of Related Objects

Disks perform rather poorly if the unit of transfer between disk and memory is small. Higher disk throughput can be achieved by reading and writing large amounts of data at a time. If certain objects are likely to be read together, these objects should be stored contiguously on disk. When a client asks for one of a group of these objects, the entire *cluster* of related objects can be read into the cache with one disk operation. This provides a very efficient implementation of prefetching.

*Research supervised by Frans Kaashoek and Barbara Liskov, and 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.

1.3 Write-Ahead Logging

The system appends all pending modifications to a persistent *write-ahead log* at transaction commit. This log is typically stored contiguously on disk and therefore all pending modifications can be made persistent with one (or at most a few) disk write.

Commit latencies are reduced even further in Thor by storing the write-ahead log in processor memory instead of on disk using a technique similar to Harp [3]. (Stability is guaranteed by replicating the in-memory write-ahead log and using uninterruptible power supplies to guard against power failures.)

1.4 Log-Structured Storage

As the write-ahead log fills up, modifications have to be moved from the log to the database storage on disk. Performance will improve if these modifications are moved to the disk with a few large writes instead of small writes of individual objects.

One technique for achieving this is to organize the entire disk as a big log [5]. When the in-memory write-ahead log fills up, its contents can be transferred to the on-disk log with big sequential writes. An important cost in such an approach is that the data stored in the on-disk log has to be periodically re-

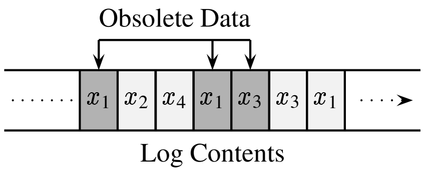

Figure 1: Log-Structured Storage on Disk.

organized to create empty disk space by reclaiming the storage occupied by obsolete data (see Figure 1).

2 Proposed Strategies

The two important performance considerations for an object-oriented database are latency and throughput. Since we use a fast replicated write-ahead log, commit latency is not a big performance issue: fetch requests are the main source of latency observed by clients. Throughput can be limited by a number of resources: processor cycles, network bandwidth, and disk-arm bandwidth. This study focuses mainly on utilizing the disk arm efficiently.

The proposed disk management strategies attempt to reduce fetch latency by efficient prefetching of objects into the cache, and improve disk arm utilization by reading and writing large chunks of data from and to the disk. An important assumption made under all of these strategies is that the set of persistent objects is partitioned into disjoint *segments*. Related objects are assumed to be clustered into the same segment. This partitioning into segments can potentially lead to better clustering on disk.

2.1 In-Place

The *in-place* strategy assigns a fixed disk location to each segment. Segments are read into the volatile cache in their entirety under the assumption that if one part of the segment is being read, then other parts will be read soon. This strategy should provide good read performance. Write performance will be improved by smart scheduling of writes that move modifications from the log to the disk.

2.2 Log-Structured

The *log-structured* strategy organizes all available disk space as a log as described in Section 1.4 and [5]. As the in-memory log fills up, its contents are appended to the disk log. This strategy has the potential to perform well on writes because the contents of the memory log can be moved to the disk log with big sequential writes. On the other hand, the contents of a segment can end up scattered over the disk and therefore read performance may suffer (see Figure 2). 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 [6].

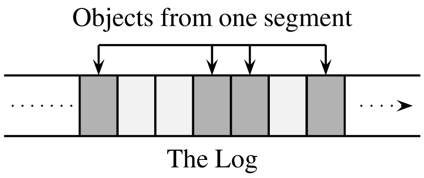

Figure 2: Segment fragmentation.

2.3 Hybrid

The *hybrid* strategy also views the entire disk as a log. However, the hybrid strategy always appends entire segments 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. 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. However this degradation in write performance may not be too severe if objects in the same segment tend to be written together. Obviously, this will depend on the effectiveness of the clustering strategy and the relation between read and write access patterns.

3 Conclusions

The proposed strategies provide trade-offs between the read and write performance of an object-oriented database. I plan to implement the three strategies as part of the Thor object-oriented database [4] and evaluate their performance under the OO7 benchmark [1]. Other benchmarks may also be used if necessary.

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 to fairly large files that are 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 [2] 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] M. J. Carey et al. The OO7 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 12–21, 1993.

[2] R. English and A. Stepanov. Loge: A self-organizing storage device. In *Winter Usenix Technical Conference*, 1992.

[3] B. Liskov et al. Replication in the Harp file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 226–38, 1991.

[4] B. Liskov et al. Distributed object management in Thor. In M. T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*. Morgan Kaufmann, 1993.

[5] M. Rosenblum and J. 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, 1991.

[6] M. Seltzer et al. An implementation of a log-structured file system for UNIX. In *Winter Usenix Technical Conference*, pages 201–220, 1993.