

A Performance Evaluation of Log-Only Temporal Object Database Systems

Kjetil Nørvåg

Department of Computer and Information Science
Norwegian University of Science and Technology

7491 Trondheim, Norway
Kjetil.Noervaag@idi.ntnu.no

Abstract

An alternative to in-place updating of data is to eliminate the database completely, and use a log-only approach. The log is written contiguously to the disk, in a no-overwrite way, in large blocks. The log-only approach is particularly interesting for transaction time object database systems (TODB), because keeping previous versions of objects is a feature that comes for free. One of the objections against log-only databases has been that the read cost will be prohibitively high because of loss of clustering. However, in our comparison of the performance of log-only and in-place update TODBs, the analysis shows that with the workload we expect to be typical for future TODBs, the log-only approach is highly competitive with the traditional in-place update approach.

1 Introduction

In most current high performance object database systems (ODBs), data is updated in-place. To support recovery and increase performance, write-ahead logging is used. This logging defers the in-place update, but sooner or later, the update has to be done. This often results in non-sequential writing of lots of pages, creating a write bottleneck.

In order to avoid the write bottleneck, it is possible to eliminate the database completely, and use a *log-only* approach. The log is written contiguously to the disk, in a no-overwrite way, in large blocks called *segments*. This is done by writing many objects, possibly from many transactions, in one write operation. This gives good write performance, but possibly at the expense of read performance, because it is more difficult to keep related data clustered. When objects are deleted or new object versions are created, the old versions become “dead”. In order to reclaim the space these dead objects occupy, the data that is still “alive” in these near empty segments can be collected and moved to a

new segment. This process, which is called *cleaning*, makes the old segments available for reuse. The cost of the cleaning is the main reason why log-only systems have not been able to achieve significant speedup compared to traditional systems. However, the fact that old object versions are not overwritten, can be utilized to realize efficient transaction-time object database systems (TODB) based on the log-only approach. The reason is that in a TODB based on traditional in-place updating, the previous version first has to be moved to the historical database, before a new version can be stored in-place in the current database. This extra cost is avoided when the log-only approach is used.

In this paper, we give a short overview of the characteristics and performance of TODBs based on the log-only approach, done in the context of the Vagabond TODB [4] which is currently under development. In an analysis based on analytical cost models, we compare the performance of a log-only TODB (LO-TODB) and a TODB based on traditional in-place update techniques (IPU-TODB). We show that with the workload we expect to be typical for TODBs, the log-only approach is highly competitive with the traditional in-place approach.

2 Related work

No-overwrite strategies have been used in shadow-paging recovery strategies earlier, e.g., in System R [1], but with the limited buffer size at that time, the performance was not satisfactory. POSTGRES [7] also employed a no-overwrite strategy, but had also its performance problems, the most important reason for this was the buffer force strategy used.

Vagabond is based on the same philosophy as log-structured file systems (LFS), which was introduced by Rosenblum and Ousterhout [5]. LFS has been used as the basis for two other object managers: the Texas persistent store [6], and as a part of the Grasshopper operating system[2]. Both object stores are page based, i.e., when an object has been modified, the whole page it resides on has

to be written back.

3 Temporal object storage

In a TODB based on the log-only approach, all object versions are stored in the log. As in all ODBs, an object is uniquely identified by an OID. An OID index (OIDX) is used to map from logical OID to the physical location of an object. In a TODB, the OIDX entries also contain the commit timestamp, and the OIDX indexes all object versions. The OIDX s can either be stored in the log, or maintained as a separate structure. In any case, OIDX entries are write-ahead logged in order to avoid forcing OIDX pages at commit time.

The granularity of data written to the log can be pages or objects. Previous log-only ODBs have been page based, and even though this works well in many contexts, it is not ideal. By operating on page granularity, you get many of the disadvantages of traditional pager servers. For example, if clustering is bad, and only a small part of a page has been updated, it is still necessary to write back the whole page. With bad clustering, main memory buffer utilization will be bad as well. A page based log-only ODB also introduces new problems to transaction management. To avoid page level locking, you essentially need to have 1) a separate log anyway, or 2) use ad-hoc techniques to solve the problem. Both solutions are likely to hurt performance and increase complexity (note that these problems do not apply to traditional systems that use in-place updating combined with write-ahead logging). One of the objections against operating on object granularity, has been that the read cost will be prohibitively high. Our performance analysis below shows that this is not necessarily true for a log-only TODB, and this fact has together with the difficulties with using page granularity convinced us that an object based log-only ODB is the way to go.

4 A comparison of performance

Our comparison of performance of the IPU-TODB and LO-TODB approaches has been done using analytical cost models. Due to space constraints, we only present a brief overview of the model and the results in this paper. A more detailed description of the analytical models and the results, using a wider range of parameters and access patterns, can be found in [3]. Our cost model is based on I/O transfer only, which is the most significant cost factor. As disk I/O is only necessary if the requested data can not be found in memory, our model also includes buffer performance.

In order to be able to recover from media failures, at least two disks have to be used. In our analysis, we assume that the IPU-TODB uses one data disk and one log disk, and that

the LO-TODB uses the two disks as a RAID 1 (mirrored) configuration.

4.1 Cost and workload models

The analytical cost functions are used to calculate the average object storage and retrieval costs. If we denote the probability that an operation is a write as P_w , the average access cost is the average of the cost of all object read and write operations: $C = (1 - P_w)C_{\text{readobj}} + P_w C_{\text{writeobj}}$. In this paper, we use speedup as a metric. If we denote the average object access time for an IPU-TODB as C_{IPU} , and the average object access time for an LO-TODB as C_{LO} , we can calculate speedup as $\frac{C_{\text{IPU}}}{C_{\text{LO}}}$. A speedup less than 1.0 means that with the given parameters an IPU-TODB will perform best, and a speedup greater than 1.0 means that an LO-TODB will perform best.

A default object size of 208 B is used in the analysis, and we assume accesses to objects in the TODB to be random, but skewed (some objects are more frequently accessed than others). We expect the hot spot in a TODB to be smaller than in a non-temporal ODB, and assume that 95% of the accesses are for 5% of the objects.

We assume the default clustering factor (the fraction of a retrieved object page that will be accessed before the page is discarded from the buffer) in the IPU-TODB to be $C = 0.2$.

4.2 Object access cost

The speedups with different amounts of main memory available for buffering and with different workloads are illustrated in Figure 1.

Object size: The average object size is an important parameter. Figure 1a shows well what can be expected, and it is important to note that even the largest object size used here, 512 bytes, is not really a very large object! The average object size is increasing as a result of new application areas and cheaper storage, which means that we can expect an even better speedup from using an LO-TODB in the future.

Update rate: In this analysis, we have used $P_w = 0.2$ as the default value for the fraction of operations being write operations. In periods, and in some application areas, we will have a higher value of P_w . This is especially the case in statistical and scientific databases, which in general experience a higher write ratio compared to other application areas. Figure 1b shows how different update ratios affect the speedup. We see that with a higher value of P_w , the speedup is considerably higher than for small values of P_w .

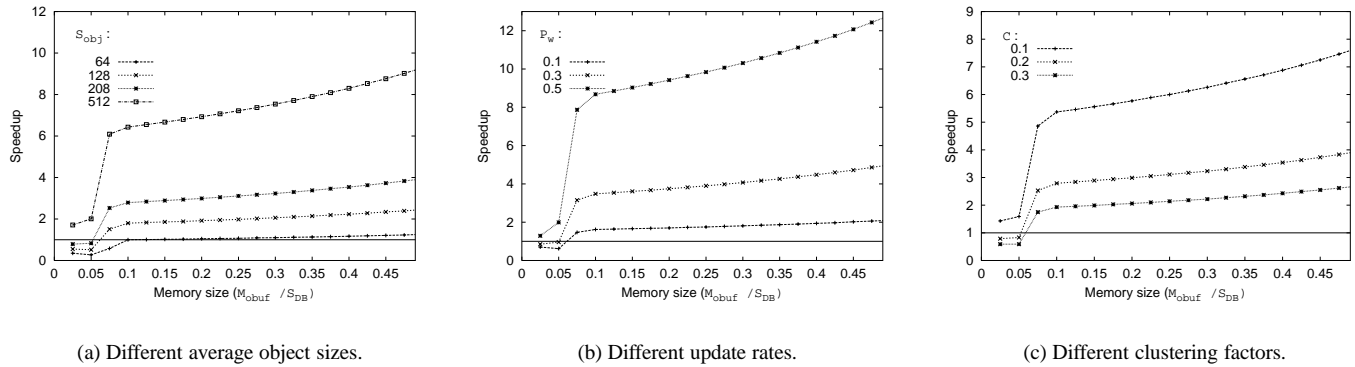


Figure 1. Speedup with different workload parameters. The memory size is given as the buffer memory size relative to database size.

Clustering: The default clustering factor in our analysis actually favors the IPU-TODB approach. Figure 1c shows how the clustering factor in IPU-TODBs affect their performance, and their relative performance to LO-TODB. This shows well how much IPU-TODBs depend on good clustering. This is an important point. In practice, with different applications accessing a database, it is difficult to achieve a high clustering factor. As is evident from the figure, if we consider a more likely clustering factor, less than 0.2, an LO-ODB will perform better than an IPU-TODB even with a much smaller amount of main memory available.

5 Conclusions

In this paper we have given a brief overview of TODBs based on the log-only approach, and have shown that with the workload we expect to be typical for TODBs, the log-only approach is highly competitive with the traditional in-place update approach. We expect the benefits of the log-only approach to be even more interesting in the future, with increasing amounts of main memory available for buffering, and increasing object sizes.

In addition to high performance, which has been the topic of this paper, the log-only approach also has other features that makes it interesting. This includes fast recovery (only one pass through the log from the last checkpoint is necessary at recovery time) and flexibility in chunk size for large objects (for example multimedia data and large multidimensional arrays, like OLAP data cubes). The log-only approach also benefits more from using RAID technology than traditional systems. The reason for this is that it writes large blocks, which is necessary to achieve high write bandwidth in RAID.

References

- [1] M. M. Astrahan et al. System R: Relational approach to database management. *ACM Transactions on Database Systems*, 1(2), 1976.
- [2] D. Hulse and A. Dearle. A log-structured persistent store. In *Proceedings of the 19th Australasian Computer Science Conference*, 1996.
- [3] K. Nørnvåg. A performance evaluation of log-only temporal object database systems. Technical Report IDI 15/99, Norwegian University of Science and Technology, 1999. Available from <http://www.idi.ntnu.no/grupper/DB-grp/>.
- [4] K. Nørnvåg. The Vagabond parallel temporal object-oriented database system: Versatile support for future applications. In *Proceedings of Norsk Informasjonkonferanse 1999*, Trondheim, Norway, November 1999.
- [5] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of the Thirteenth ACM Symposium on Operating System Principles*, 1991.
- [6] V. Singhal, S. Kakkad, and P. Wilson. Texas: An efficient, portable persistent store. In *Proceedings of the Fifth International Workshop on Persistent Object Systems*, 1992.
- [7] M. Stonebraker. The design of the POSTGRES storage system. In *Proceedings of the 13th VLDB Conference*, 1987.