An Approach to High-Performance Scalable Temporal Object Storage

Kjetil Nørvåg

Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
email: noervaag@idi.ntnu.no

Abstract. In this paper, we discuss features that future database systems should support to deliver the required functionality and performance to future applications. The most important features are efficient support for: 1) large objects, 2) isochronous delivery of data, 3) queries on large data sets, 4) full text indexing, 5) multidimensional data, 6) sparse data, and 7) temporal data and versioning. To efficiently support these features in one integrated system, a new database architecture is needed. We describe an architecture suitable for this purpose, the Vagabond Temporal Object Database system. We also describe techniques we have developed to avoid some potential bottlenecks in a system based on this new architecture.

1 Introduction

The recent years have brought computers into almost every office, and this availability of powerful computers, connected in global networks, has made it possible to utilize powerful data management systems in new application areas. The increasing performance and storage capacity, combined with decreasing prices, has made it possible to realize applications that previously were too heavy for current computer hardware.

High performance and storage capacity is not necessarily enough. We need support software, e.g. database systems, operating systems, and compilers, able to benefit from current *and future* hardware. This often means rethinking previous solutions, similar to what was done in the hardware world with the introduction of the RISC concept.

In this paper, we will concentrate on database systems, quite likely to be the bottleneck in many future information systems if not adequately designed. The first step in the process of rethinking old solutions has already been done, with the advent of *object database system* (ODBs). While relational database systems (RDBs) have good performance for many of the previous, traditional, application areas, new applications demands more than RDBs can deliver. The increased modeling power and removal of the language mismatch in ODBS, has made integration between the application programs easier, and in many cases helped to increase the application performance.

Previously, data has lived in an artificial, modeled world after they had been inserted into the database. This created a mismatch in many ways similar to the language mismatch. What we would like, is that database systems should support a world more similar to our own, which includes *time and space*. This is not at all a new observation, especially the aspects of temporal database management have been an active research area for many years. However, current database architectures, adequate for yesterday's applications, will have problems coping with tomorrow's application. In this paper, we describe a new architecture, more suitable for tomorrows applications, the *Vagabond Temporal Object Database System*. We give an overview of Vagabond, and describe some of the new techniques we have developed to make Vagabond able to deliver the high performance and scalability needed for future applications.

The organization of the rest of the paper is as follows. In Section 2 we give an overview of related work. In Section 3, we describe some application areas that have only limited support in existing database system. Based on this discussion, we summarize the features future database systems should support, and describe assumptions and features that motivate the design of the Vagabond system. In Section 4 we discuss some techniques that can increase the performance of a temporal ODBs. Finally, in Section 5 we conclude the paper.

2 Related Work

Log-structured file systems, whose philosophy the log-only approach of Vagabond is based on, was introduced by Rosenblum and Ousterhout [7]. LFS has been used as the basis for two other object managers: the Texas persistent store [8], and as a part of the Grasshopper operating system[1]. Both object stores are page based, i.e., operations in the database are done on page granularity. To our knowledge, there has been no publications on other object LFS based log-only ODBs.

3 The Need for a New Architecture

When designing new database systems, it is important to study the current as well as possible future applications of the system. We can categorize application areas into *existing* application areas, and *emerging* application areas. Existing application areas includes the traditional database areas, like typical transaction processing, well suited for RDBs, and application areas where application specific database systems or file systems have been used earlier, because current general purpose database systems can not handle the performance constraints. Emerging application areas include new application areas, that are emerging as a response to the increased computer performance in general, as well as application areas that are a response to other technologies, e.g., World Wide Web.

Examples of existing applications, where database systems until recently have been a potential performance bottleneck include geographical information systems, scientific and statistical database systems, and multimedia systems. Examples of applications where increased database support will be needed to deliver the desired performance include temporal database systems and XML/semistructured data management. Based on the characteristics of these application areas, we have identified some features that we believe future systems should support:

- Efficient support for large objects.
- Isochronous delivery of data.
- Queries on large data sets.
- In applications where low update rates appear, this should be exploited to increase performance.
- Support for full text indexing.
- Support multidimensional data.
- Support sparse data, for example by the use of data compression.
- Dynamic clustering and dynamic tuning of system options and parameters.
- Temporal data support/version management.

Until now, no single system has supported all these features. For some of the features listed, ad-hoc solutions exists, but these are often not scalable, or will not work well together with support for the other features. We think that future systems should support these features, in *one integrated system*. This is the goal of the Vagabond project. Vagabond is designed to support the listed features, with a philosophy based on the following assumptions:

1. Although many of the current problem can be handled by future main memory database systems (MMDBs), there are many problems (and more will appear, as the computers become powerful enough to solve them) that are too large to be solved by a MMDB alone. However, the size of main memory increases fast, and it is very important to utilize the available main memory as much as possible to reduce time consuming secondary memory accesses.
2. *The main bottleneck* in a database system for large databases is still secondary memory access. In a database system, most accesses to data are read operations. Consequently, database systems have been *read optimized*. However, as main memory capacity increases, the amount of disk write operations relative to disk read operations will in general increase. This calls for a focus on *write optimized* database systems.
3. To provide the necessary computing power *and* data bandwidth, a parallel architecture is necessary. A shared-everything approach is not truly scalable, so our primary interest is in ODBs based on shared-nothing multicomputers. With the advent of high performance computers, and high speed networks, we expect multicomputers based on commodity workstations/servers and networks to be cost effective.
4. In many application areas, there is a need for increased data bandwidth, not only increased transaction throughput (although these points are related). This is especially important for emerging application areas that have a need for high data bandwidth. Examples are video on demand, and supercomputing applications, which have earlier used file systems because database systems have not supported delivery of large data volumes.
5. Even though set based queries have been a neglected feature in most ODBs, we expect it to be just as important in the future for ODBs as it has been previously for relational database systems. The popularity of the hybrid object-relational systems justifies this assumption.
6. Distributed information systems are becoming increasingly common, and they should be supported in a way that both facilitates efficient support for distribution, *and* efficient execution of local queries and operations.

We will now describe the architecture and some interesting aspects of the Vagabond ODB.

3.1 Log-Only Storage

In most current database systems, write ahead logging (WAL) is employed to increase throughput and reduce response time. WAL defers the non-sequential writing, but sooner or later, the data has to be written to the database. This often results in the writing of lots of small objects, almost always one disk access for each individual object. Our solution to this problem, is *to eliminate the current database completely*, and use a *log-only* approach, similar to the log-structured file system approach [7]. The log is written contiguously to the disk, in a no-overwrite way, in large blocks. This is done by writing many objects and index entries, possibly from many transactions, in one write operation. This gives good write performance, but possibly at the expense of read operations.

Already written data is never modified, new versions of the objects are just appended to the log. Logically, the log is an infinite length resource, but the physical disk size is, of course, not infinite. We solve this problem by dividing the disk into large, equal sized, physical segments. When one segment is full, we continue writing in the next available segment. As data is vacuumed, deleted or migrated to tertiary storage, old segments can be reused. Deleted data will leave behind a lot of partially filled segments, the data 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. By combining cleaning with reclustering, we can get well clustered segments. In a traditional system with in-place updating,

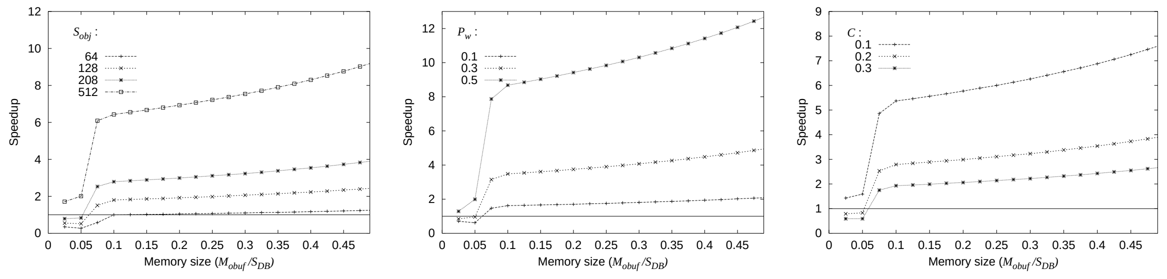

Figure 1. Speedup with different workload parameters: different object sizes to the left, different update rates in the middle, and different clustering factors to the right. The memory size is given as the buffer memory size relative to data base size.

keeping old versions of objects, which is required in a transaction-time temporal database system, usually means that the previous version has to be copied to a new place before update. This doubles the write cost. In Vagabond, this is not necessary. Keeping old versions comes for free (except for the extra disk space). Thus, our system supports transaction-time temporal database systems in an efficient way.

Because each new version of an object is written to a new place, logical object identifiers (OIDs) are needed. When using logical OIDs, an OID index (OIDX) is needed to do the mapping from logical OID to physical location when retrieving an object. The index entries in the OIDX, the *object descriptors* (OD), contains the physical address for an object, and in a transaction-time temporal ODB (TODB), the timestamp as well. In a traditional non-temporal ODB, the OIDX needs only be updated when objects are created, not when they are updated. In a log-only ODB, however, the OIDX needs to be updated on every object update. This might seem bad, and can indeed make it difficult to realize an efficient non-temporal ODB based on this technique. However, in the case of a TODB, the OIDX needs to be updated on every object update also in the case of in-place updating, because either the previous or the new version must be written to a new place. Thus, when supporting temporal data management, the indexing cost is the same in these two approaches.

Our storage structure is very well suited as a basis for a temporal database system. We never overwrite data, so keeping old versions comes for free. We maintain the temporal information in the index, which makes retrieval efficient, without an additional index. We have done an analysis of the possible speedup when using the log-only approach instead of in-place updating. Figure 1 illustrate the speedups with different amounts of main memory available for buffering, using an 95/05 access pattern, and with different workload parameters.

To the left in Figure 1 we see the speedup with different average object sizes. As can be expected, the object size is an important parameter, and the gain increases with increasing object sizes. 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 log-only TODB in the future.

In our analysis, we assumed an update rate of $P_w = 0.2$ 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. In the figure, the speedup with an average object size of 208 bytes and different values of P_w is illustrated.

We have assumed an average clustering factor (the fraction of a retrieved object page that will be accessed before the page is discarded from the buffer) in the in-place update based TODB to be $C = 0.2$. In practice, this value will be often be smaller. As Figure 1 illustrates, the speedup in that case is much higher. A more detailed description of the analytical models and the results, using a wider range of parameters and access patterns, can be found in [4].

3.2 Parallelity and Distribution in Vagabond

The Vagabond architecture is a system designed for high performance, and one strategy to achieve this, is to base the design on the use of parallel servers. Data is declustered over a set of servers, which we call a *server group*. It is possible to add and remove nodes from the confi guration. The servers in a server group will cooperate on the same task. In this way, it is possible to get a data bandwidth close to the aggregate bandwidth of the cooperating servers. To benefi t from the use of a parallel server groups, it is supposed that the servers in one server group are connected by some kind of high speed communication.

In many organizations, it is also desirable to have the data in a *distributed system*, and the demand for support of distributed databases is increasing. To satisfy this, we use a hybrid solution: *a distributed system, with server groups*. The connections between the server groups in the distributed system have in general less bandwidth than the connections between the servers in a server group. Objects are clustered on server groups based on locality as is common in traditional distributed ODBs, but one server group can contain more than one computer (a kind of “super server”). Objects to be stored on a server group are declustered on the servers in the group according to some declustering strategy, e.g., hashing.

3.3 Objects in Vagabond

In our storage system, all objects smaller than a certain threshold, e.g., 64 KB, are written as one contiguous object. They are not segmented into pages as is done in other systems) Objects larger than this threshold are segmented into *subobjects*, and a *large object index* is maintained for each large object. Parts of the object can reside on different physical devices, possibly on different levels in the storage hierarchy.

The value of the threshold can be set independently for different object classes, something which is very useful, because different object classes can have different object retrieval characteristics. Typical examples are a video and a general index. In a video, you want to retrieve one large block of the video each time, it is needed to play the video. When searching an index however, often relatively small nodes are desired. Similar for both video and index retrieval is that you only want a small part of the object. In other situations, e.g., retrieval of an image, you want to display the image, and therefore want to retrieve the whole image at once.

4 Removing Bottlenecks

During the design of Vagabond, we have used cost modeling to identify potential bottlenecks in the proposed system, and to fi nd techniques to avoid them. As can be expected, OIDX management is potentially very costly. To reduce the indexing cost, we have developed a new OIDX structure for temporal ODBs [5], as well as several novel techniques that reduce the cost of OID indexing:[1]

- “Writable” object descriptor cache for temporal ODBs [6].
- Persistent caching of OIDX entries [3].

[1] It is important to note that the OIDX bottleneck problem also exists in a TODB based on traditional page server/in-place update techniques. Even though in a traditional *non*-temporal ODB, the OIDX only needs to be updated when objects are created, in a TODB, each object update creates a new version, and the OIDX needs to be updated. This means that the proposed techniques are applicable to all TODBs, not only log-only systems.

A second bottleneck is object retrieval. The Vagabond system is write optimized, and as a result, object retrieval and index lookup can become a serious bottleneck. We will use some techniques to reduce the number and size of read operations, which can improve object retrieval performance considerably, with only marginal write penalties:

- The use of signatures in the OIDX [2].
- Object compression. With a log-only approach, objects are written to a new location every time, so that *we only use as much space as the size of the current version written.* In a system employing in-place updates, it is diffi cult to benefi t from object compression, because the compression ratio will very from version to version, and it is diffi cult to know how much space to reserve.

Compression will also improve write effi ciency, as it reduces the amount of data needed to be written to disk. We will now give an overview over the techniques. For a more detailed description, we refer to the corresponding papers where these techniques are presented and analyzed [2, 3, 6].

5 Conclusions

In this paper, we have discussed features that future database systems need to support to deliver the required functionality and performance to future applications. In order to be able to support these features, new database architectures ares needed to make effi cient and scalable systems. In this paper, we have described an architecture suitable for this purpose, the Vagabond Temporal Object Database systems, which is currently under development at the Norwegian University of Science and Technology. In order to avoid potential bottlenecks in the case of very large databases, new techniques to reduce object identifi er indexing cost and object read cost are desired. We have described several such techniques, including a writable OD cache, the persistent cache, a new object identifi er index, and the use of object signatures integrated into the object identifi er index.

References

1. D. Hulse and A. Dearle. A log-structured persistent store. In *Proceedings of the 19th Australasian Computer Science Conference*, 1996.
2. K. Nørvåg. Efficient use of signatures in object-oriented database systems. In *Proceedings of Advances in Databases and Information Systems, ADBIS'99*, 1999.
3. K. Nørvåg. The Persistent Cache: Improving OID indexing in temporal object-oriented database systems. In *Proceedings of the 25th VLDB Conference*, 1999.
4. K. Nørvåg. A performance evaluation of log-only temporal object database systems. Technical Report IDI 15/99, Norwegian University of Science and Technology, 1999.
5. K. Nørvåg. The Vagabond temporal OID index: An index structure for OID indexing in temporal object database systems. In *Proceedings of the 2000 International Database Engineering and Applications Symposium (IDEAS)*, 2000.
6. K. Nørvåg and K. Bratbergsengen. Optimizing OID indexing cost in temporal object-oriented database systems. In *Proceedings of the 5th International Conference on Foundations of Data Organization, FODO'98*, 1998.
7. 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.
8. 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.