Parallel Object Server Architecture and Performance

Petr Kroha and Joerg Lindner

Fakultät für Informatik, TU Chemnitz
09107 Chemnitz, Germany
kroha@informatik.tu-chemnitz.de

Abstract. In this paper we describe the architecture and the performance of an implemented prototype of a parallel object server that we developed for storing fine grained objects. Finally, we present results of our experiments that concern the question how the speedup depends on the number of processors involved and the number of users working simultaneously.

1 Introduction

Sequential computers that are currently used as servers have their physical limits of performance. It is difficult to estimate the time point when their limits will be reached, but the usage of parallelism in parallel and distributed systems may be the only possibility to increase the performance of servers in the future.

Some applications, especially in engineering, use servers for management and storing fine grained objects. For storing and processing complex, fine grained data, the object-oriented DBMSs seem to be in principle more suitable than the relational DBMSs [2]. The normalization of tables in the relational model can cause the splitting of fine structured objects into many tables. During the phase of fetching objects, many tables are to be joined for getting the parts of objects together again [5]. This is paid by less performance.

The problem is that currently there are in principle no object-oriented DBMSs for parallel computers available as commercial products which could be used and tested. Because of the lack of standard, large variety of parallel computers, and a small demand, their arrival cannot be expected in the near future. Well, there is an object-relational ORACLE implementation on nCUBE, but object-relational is not object-oriented and nCUBE is only one of hundreds of types of parallel computers.

In this paper we describe our prototype of an object-oriented server OPAS (Object-oriented PArallel Server) running on a parallel computer (shared disk, distributed memory) as a data repository for a CASE tool. For our experiments, we have used data of two CASE tool applications and have measured speedup by changing the number of processors, the number of users, and other parameters.

The rest of this paper is organized as follows. Related work will be described in Section 2. In Section 3 we explain what advantages hierarchical data structures may have for parallel processing. Section 4 briefly describes the software architecture designed for the parallel object server. In Section 5, we discuss the implementation of the prototype on a parallel computer having shared-disk and distributed memory architecture. The results achieved and future work are discussed in Section 6.

2 Related work

The research of OODBMSs in a parallel environment seems to be at its very beginning. Until now, the OODBMS producers have paid very little attention to the parallel environment until now because of the small market.

In [8] we described how data have been stored in our first prototype, how buffers are organised, and how efficiently the object manager, the lock manager, and the disk manager have been working. Specific questions concerning properties of data repositories of CASE tools used in a multi-user environment in software engineering have been discussed in [9].

We did not find any papers discussing object-oriented parallel servers. Thus, we cannot compare our results with other results achieved by other researchers. The key contribution of this paper is that this is the very first attempt in this research direction.

3 Data structures and parallel processing

For the first attempt we simplified and focused our considerations on two groups of hierarchical data structures stored in the data repository:

- Structured data representing more levels of abstraction, e.g. stored in trees, are naturally hierarchically organized. They are typical in top-down analysis and bottom-up synthesis.
- Structured data representing only one level of abstraction, e.g. stored in lists and graphs, can be seen for purposes of parallel processing as hierarchical structures having a root (containing the name of the whole structure and OIDs of elements of the structure) and only one level of nodes which are leaves. Such structures will be used e.g. for representing ER-diagrams, finite state machines, Petri nets, etc.

From the point of view of data structures (objects), there are the following sources of parallelism:

- Object components which have a common father can be searched in parallel. This kind of parallelism will be called intra-object parallelism.
- Objects of different users can be searched in parallel. This kind of parallelism will be called inter-object parallelism.

To get data for our experiments we considered the development of two applications (hotel, warehouse). The top-down method of stepwise refinement via decomposition delivered hierarchically organized data.

In contrast to multimedia applications, large objects in CASE tools data repositories are usually aggregated and contain OID's of their components. Basic components contain their type, some parameters, and some OID's of related basic components if any. Our test applications generated objects of size between 50 and 500 bytes having the following statistical distribution: 48 % of objects of size 50-100 bytes, 30 % of objects of size 101-150 bytes, 11 % of objects of size 151-200 bytes, and 10 % of objects of size 201-500 bytes.

In our test examples, data representing data flow diagrams have been used, but we believe that similar results could be achieved for other hierarchically structured data, e.g. for relations Is-A (inheritance) and Is-Part-Of (aggregation) used in object-oriented modeling. As stated above, frequently used ER-diagrams can be processed as one-level hierarchical structures.

4 Hardware and software architecture

The parallel computer we used has the shared-disk architecture and a distributed memory. All accesses of its 128 processors to its shared permanent memory must be synchronized for using 8 access channels. Although it was originally bought for another purpose, we tested its features in our project and we found it being suitable for the given purpose as we describe in conclusions.

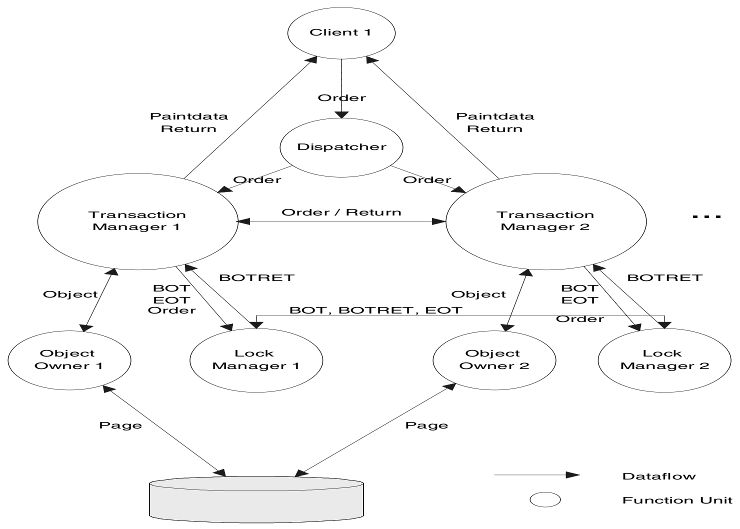

Fig. 1. Architecture of parallel server

The software architecture of the parallel server can be seen in Figure 1. The interface between the object server and its environment will be represented by a dispatcher. It communicates with clients, accepts their queries, and asks some of transaction managers to execute the corresponding tasks. The load of transaction managers will be planned to be evenly distributed.

The transaction manager concept has been designed as distributed, i.e. as a set of transaction managers. This makes possible the parallel processing of transactions (Fig. 2).

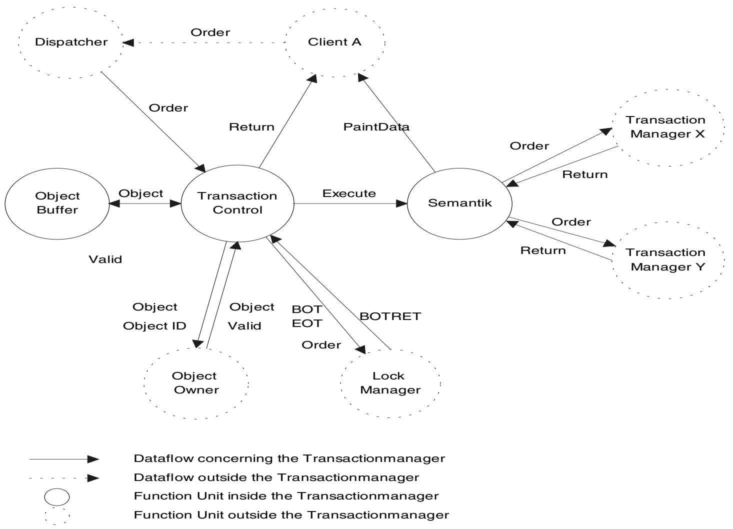

Fig. 2. Transaction manager

Because of possible problems of redundancy, the concept of an object owner has been designed. An object owner is responsible for the consistency of some subset of objects, i.e. there is a set of object owners responsible for the consistency of all data. Each object owner has available its own page-oriented disk manager for accessing data on files. The disk manager has its own page buffer. Because of the dynamic assignment between objects and transaction managers, replication of objects in the buffer can exist. The object owners get information about current values of object attributes from the transaction managers and check the consistency of their objects, i.e. of objects they are responsible for. Object owners represent an interface between transaction managers and disk managers.

We used a distributed lock manager, i.e. a set of lock managers. They are responsible for the synchronization of parallel transactions. Each lock manager is assigned a subset of objects. The distributed locking process concerns all corresponding lock managers. The simple 2-phase protocol has been used for locking to avoid deadlocks in the locking phases. Each lock manager uses a lock

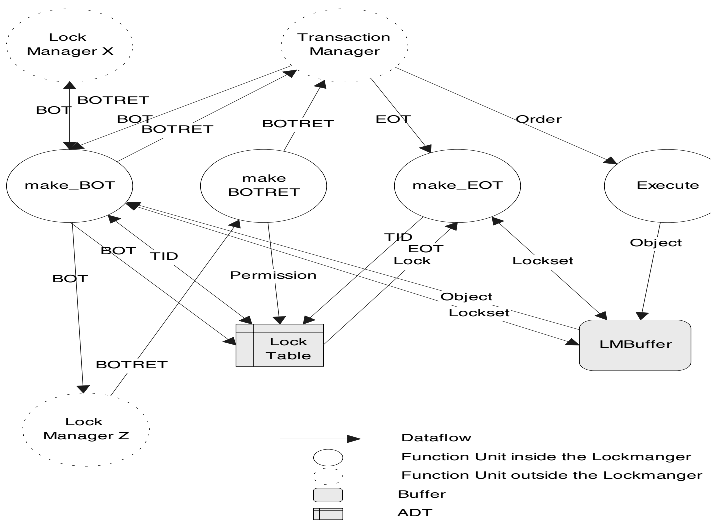

Fig. 3. Lock manager

table (Fig. 3) into which all information about locking and unlocking concerning its subset of objects will be written. However, the probability of such access conflicts seems to be very small, because software development team members normally work on disjunct objects. If not, versioning should be used.

Each transaction manager checks the consistency of objects by means of the corresponding object owner (Fig. 4) and controls the synchronisation by means of the lock managers.

The processing of a client's contract is done as follows. The dispatcher accepts the contract (query), which has the form of a message that is to be sent to an object Obj. According to the load distribution of transaction managers, the dispatcher finds a suitable transaction manager and delegates the contract to it. This transaction manager starts a new transaction by asking the lock manager (responsible for the object Obj) to start the locking process. Then, it tries to lock all involved objects of the intended transaction. While the lock manager is running, the transaction manager looks for the object Obj. If it is in its object buffer, the object owner will be asked for its validity. If the object Obj is not in the object buffer or if its value is obsolete, the transaction manager asks the object owner for an updated copy. Having fetched the object, the transaction manager asks the corresponding lock manager about the result of the locking process. If at least one object has just been locked by another transaction, later, the access attempt will several times be repeated until the desired object Obj is locked for the planned transaction.

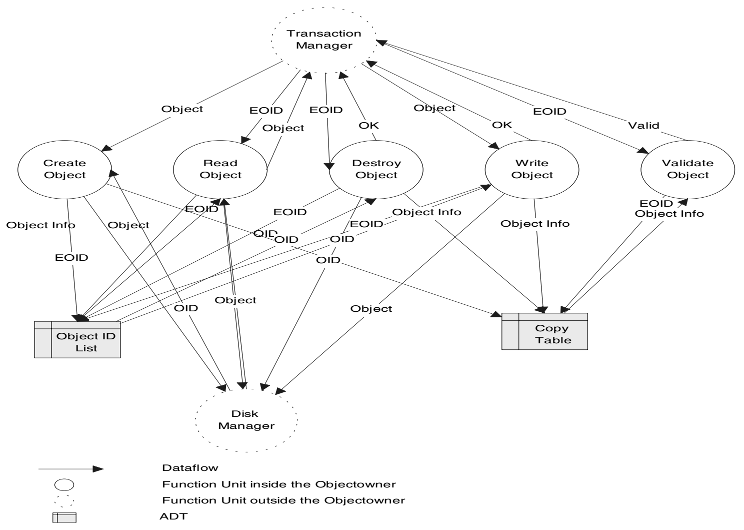

Fig. 4. Object owner

Now the message can be delivered to the object Obj which can cause some subtransactions to be started. After a transaction has been successfully finished, the lock managers unlock the objects involved. If some values of attributes have changed, the corresponding object owners will obtain the updated versions of these objects.

Transactions will be constructed using concepts of intra-object parallelism and inter-object-parallelism. As long as there are no synchronization conflicts in processing transactions and as longs as there are enough nodes, transactions are running fully in parallel. Synchronization conflicts occur, if some transactions ask for the same objects. Lock managers will solve them, but this will reduce the degree of overlapping of different queries, of course. However, on real conditions, although users are working on the same project using the same data repository, they are usually not working simultaneously on the same object in software engineering. If they exceptionally would, the process would slow down. In our test data, sets of objects involved in each transaction were disjunct. Thus, restricting factors in this sense are the number of processors and the accesses (read/write conflicts) to the shared disk.

5 Implemented prototype OPAS

In the first step, we simulated the logical concepts of the software architecture and the functionality of the server. We used Ada95 on a Sun machine for writing the first version of OPAS as a running design specification. We implemented clients, a dispatcher, transaction managers, object owners, lock managers, and disk managers as tasks. To describe the shared-disk architecture, we wrote a disk task. All object owners have to share this disk task, which has exclusive access. Queries of clients have been written into batch files. This has been done to simplify the subsequent debugging of the prototype in C++.

This step helped us to solve some problems. Besides proving the algorithms we obtained an overview about the dependence between the speedup and the number of processors used for a single-user and for a multi-user environment. We also found that the dynamic allocation of functional units which has been implemented as a part of the dispatcher in the first version does not work efficiently enough. Therefore, we have used the static allocation in the further development of the prototype. The use of the dynamic strategy would be suitable if the methods of the data model enclosed a large amount of instructions. CASE tools for software engineering seldom have such methods, but CAD tools used in other engineering disciplines have them very often.

After this first phase, the prototype was written in C++ for the parallel computer PARSYTEC/GC Power Plus - 128 with operating system PARIX. Our configuration of this computer consists of 64 nodes organized into 2D lattice.

The transfer unit of our server is one page and the optimal possibility for data transfer would be the one-to-one mapping between objects and pages. This is not always possible because some objects are (or can become) bigger than one page. We used a mapping between one object and one or more pages. After experiments with data used in CASE tools [8], the page size was defined 800 bytes because our objects mostly fit to this size.

We have placed the transaction manager and object owner on one processor and the lock manager on the next processor in the node. Under PARIX, more than one thread can be started in one node in the context of the node's program. If more than two threads are used in one node, time-sharing will be used.

6 Results, conclusions, future work

As a conclusion, we present some speculative results concerning the used hardware architecture and some experimental results concerning the behavior of our prototype in response to the experimental data.

First, we argue that the shared-disk hardware architecture used in our project does not bring significant disadvantages when used as hardware background for parallel object-oriented databases as data repositories for fine grained objects.

Usually, the shared-nothing hardware architecture will be recommended as the more suitable one for relational database systems [1], but when using it for our purposes in client/server environment, an important question is where the common data repository should be stored:

- If the common data repository is stored distributed and non- replicated on many disks then the responsible processor has to ask other processors to access and send data while processing the query. This increases the overhead.

- If the common data repository is stored distributed and replicated on many disks then it would bring an overhead (because of the checking of consistency) depending very strongly on how often the data will be used and changed.
- If the common data repository is stored on one disk then we have the same problem as in the shared-disk architecture.

When using the shared-disk architecture users are competing for the disk. In a trivial case, there can be only one disk connected. Usually, there are more channels in the switch which can work in parallel and support more disks. The query GET OBJECT will be processed by one processor which does not have sole access to the data, i.e. sometimes it has to wait for a free channel, but it can access to the data directly, i.e. without asking other procesors for help. It seems to be a good solution that the shared, common data repository is stored on the shared disk.

Second, in our experiments with the implemented prototype we investigated at first how the speedup depends on the intra-object parallelism in a single-user environment.

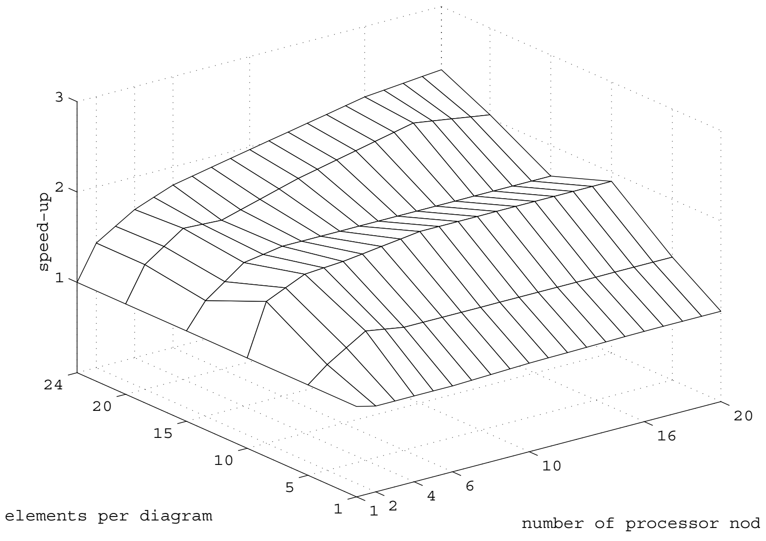

Fig. 5. speedup = f1(elements, nodes)

The function f1 (Fig. 5) shows the influence of the data structure of our objects on the speedup for increasing number of processor nodes involved. The number of elements per diagram (components of an aggregated object) represents the degree of the intra-object parallelism of the investigated data model. If an object is to be fetched and shown, its data structure has to be synthesized using pointers to its components that will be searched in parallel. We can see that the use of a parallel machine in this direction would not bring too much (factor 2.5 for 24 elements in diagram and 20 nodes used), at least not too much for objects composed from 1 to 24 components which we have tested. Of course, it depends strongly on the cardinality of aggregation used in classes which are typical for the specific application. For some CAD tools this kind of parallelism can bring perhaps much more speedup.

In the multi-user environment we investigated the dependence of the speedup on the number of clients and number of nodes. However, as shown above, there are some additional dimensions expressing the properties of data namely the size of methods and the number of components. Thus, it can be drawn in 3D only using some specific cuts, i.e. additional conditions. Fig. 6 represents the function f3 in the situation when all clients want to paint an object (not the same) consisting of 20 components 10 times (properties of the painting method). The situation when all clients want to process disjoint objects is a typical situation for using CASE tools and we used it in our tests.

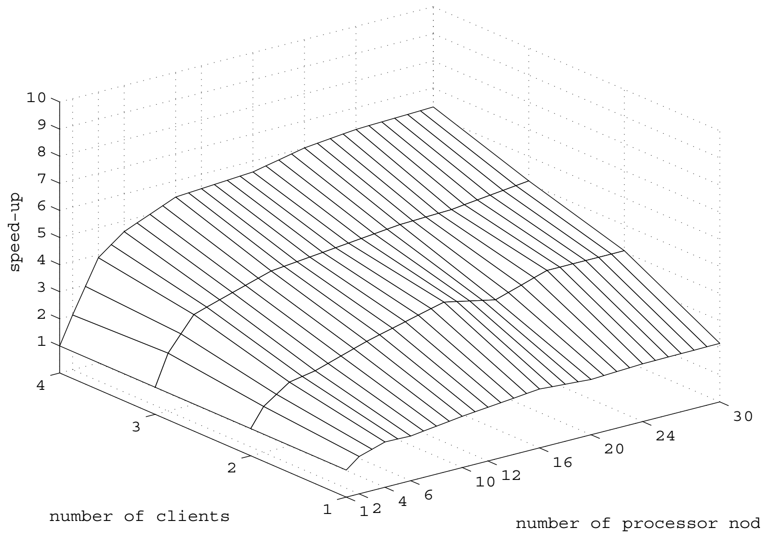

Fig. 6. speedup = f3(clients, nodes) for the painting method

Our main goals for the future research will concern the extension of test data and the usage of alternative concepts. Using extended test data, the implemented prototype will be tested in the future with the goal to investigate other dimensions of this problem given by improved strategies of parallel and distributed processing, e.g. optimistic locking.

References

1. De Witt, D., Naughton, J.F. et al: ParSets for Parallelizing OODBMS Traversal: Implementation and Performance. In: Proc. 3rd International Conference on Parallel and Distributed Information Systems, pp. 111-120, Austin, September 1994.
2. Emmerich, W., Kroha, P., Schäfer, W.: Object-Oriented Database Management Systems for Construction of CASE Environments. In: Marik, V. et al. (Eds.): Proceedings of the 4th Int. Conference DEXA'93, Lecture Notes in Computer Science, No. 720, Springer, 1993.
3. Freitag, B., Jones, C.B., Lengauer, Ch. Schek, H.-J.: (Eds.): Object Orientation with Parallelism and Persistence. Kluwer Academic Publishers, 1996.
4. Kroha, P.: Translation of a Query in OODBMS into a System of Parallel Tasks. EUROMICRO'92, Microprocessing and Microprogramming 37 (1993), North- Holland.
5. Kroha, P.: Objects and Databases. McGraw-Hill, 1993.
6. Kroha, P.: Shortcoming and Extensions of Relational DBMS. In: Adelsberger,H. et al. (Eds.): Information Management in Computer Integrated Manufacturing. Lecture Notes in Computer Science, No. 973, Springer, 1995.
7. Kroha, P.: Softwaretechnologie. Prentice Hall, 1997. (In German)
8. Kroha, P., Rosenbaum, S.: Object Server on a Parallel Computer. In: Wagner, R.R. (Ed.): Proceedings of the 8th International Workshop on Database and Expert Systems Applications DEXA'97, IEEE Computer Society, Toulouse 1997.
9. Kroha, P., Lindner, J.: Parallel Object Server as a Data Repository for CASE Tools. In: Croll, P., El-Rewini, H.(Eds.): Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems PDSE'99, Workshop of ICSE'99, pp. 148-156, IEEE Computer Society, Los Angeles, May 1999.
10. Lindner, J.: Properties of a Parallel Object Server as a Data Repository for CASE-Tools. M.Sc. Thesis, Faculty of Informatics, TU Chemnitz, 1998. (In German).
11. Maier, D.: Making Database Systems Fast Enough For CAD. In: Kim, W., Lochovsky, F. (Eds.): Object-oriented Concepts, Databases and Applications, pp. 573-582, ACM Press 1989.
12. Radestock, M., Eisenbach, S.: An Object Model for Distributed and Concurrent Programming Based on Decomposition. In: [3].
13. Stonebraker, M.: The case for shared nothing. Database Engineering, Vol. 9, No. 1, 1986.
14. Valduriez, P.: Parallel database systems: the case for shared nothing. In: Proc. of the 9th Int. Conf. On Data Engineering, pp. 460-465, Vienna 1993.