Performance Comparison of Distributed Object Server Implementations

Petr Kroha[1], Mathias Kurth[2], and Michael Fleischer[1]

[1] Dept. of Information Systems, Faculty of Informatics, Technical University Chemnitz, Strasse der Nationen 62, 09111 Chemnitz, Germany
kroha@informatik.tu-chemnitz.de
[2] Department of Computer Science, Humboldt University Berlin, 10099 Berlin, Germany
kurth@informatik.hu-berlin.de

Abstract. In this paper we present performance measurements in a cluster environment. First, we briefly explain our version of optimistic concurrency control and load balance. Then we compare performance and speed-up of a CORBA- and an AspectC++-implementation of a distributed object server for pessimistic and optimistic concurrency control.

Keywords: Distributed processing, cluster environment, object server, aspect-oriented application, performance of cluster applications

1 Introduction

Object-oriented databases represent an implementation tool for data repositories of some specialized applications, which often process not only selected attributes of objects but complete objects. When using relational databases, attributes of many objects can be processed very quickly if they are stored as columns in one table. Processing of complete objects stored in a relational database means to synthesize objects from many tables where they are spread because of the necessary normalization of relations. There is a rule of thumb that if it is necessary to join more than six tables to answer a query, an object-oriented database could bring advantages [4].

CASE tools are usually built as a chain of editors that have a common data repository, in which all input and output data of editors are stored as fine-grained objects [1]. We have built a CASE tool with an object-oriented data repository and have found that its performance is not sufficient especially for multi-user environment. This was our motivation to investigate possibilities of parallel and distributed object servers. In the next step we built an object server for the parallel computer Parsytec and have analyzed its performance [6]. Thereafter we switched from a parallel to a cluster environment, improved the used software architecture and algorithms to increase performance, and used various implementation technologies (AspectJ, AspectC++, CORBA) because of the planned porting. We supposed that optimistic concurrency control can bring us big advantage because semantic features of our application mean that conflicts and roll-backs should be only rare exceptions. We wanted to know how much we can win by using optimistic concurrency control.

The rest of the paper is organized as follows. In section 2 we discuss related work. In section 3 we briefly explain which cross-cutting concepts of concurrency we modelled as aspects in our proposed and implemented optimistic concurrency control method. In section 4 our new version of the optimistic concurrency control for cluster environment is presented. Section 5 describes our approach to load balancing. In the last section we present some measurement results and conclude.

2 Related work

Empirical comparisons between optimistic and pessimistic concurrency control have been published in [8] and [3]. The former compares both approaches with respect to shared disk systems. The outcome is that the lock protocol outperforms the optimistic approach in terms of performance but the performance strongly depends on a centralized or distributed realization. On the other hand, the optimistic protocol scaled up better than the pessimistic one. The latter concludes that an optimistic approach could be a better choice for object-oriented database systems in terms of performance and scalability, especially in client-server environments.

We have not found quantitative measurement results that would be relevant for our shared-nothing cluster system that should work as a data repository of a CASE tool.

The applicability of aspect-oriented techniques to database systems is known and was already investigated, e.g. in [9]. We aspectized concurrency control from the main program. Our goal was to support portability of our application. In addition, it was simply possible to implement not only the optimistic but also the pessimistic concurrency control, which helped us to compare performance of both methods. It is another indication for the broader applicability.

3 Aspects and Concurrency Control in OODBMS

The term aspect-oriented programming (AOP) covers approaches for reaching separation of concerns.

We concentrated on concurrency control as a cross-cutting concern at DBMS-level. For most systems the dominant dimension of decomposition is transaction processing and resource management. This leads to a scattering of code related to concurrency control.

In OPAS (Object-oriented PArallel Server [5], [6]), the optimistic concurrency control (OCC) crosscuts the Transaction Manager (TM). The interactions between TM and OCC are as follows. During transaction processing the TM acquires a set of resources, namely objects in our case. It processes orders on acquired objects, which may read or change it. The start and end time of an object access and the corresponding access mode (read,write) must be noted and handed over to the OCC in order to validate the access. If the validation fails, the transaction must be aborted.

Using AOP both TM and Validation Manager (VM, conceptual component for OCC) could be implemented in a separate manner and then put together by the weaver AspectC++ in the preprocessing phase. An aspect captures the acquisition of an object

through an advice and stores the start time of access. The completion of an object access is intercepted, too. The corresponding advice (join-point model of AOP) determines the end time and access mode. It calls the VM for validation using the collected information and processes the validation result. We do not explain aspect-oriented programming in more details because it is not the main topis of this paper.

4 Optimistic Concurrency Control

Isolation is one of the ACID properties. It protects concurrently running transactions from seeing each others' uncommitted data, which could lead to inconsistency. Serializability is the correctness criterion for isolation. It states that the concurrent execution of several transactions must be equivalent to at least one serial execution [?].

A common approach to isolation is locking, which basically means that locks are granted to transactions to protect a resource from concurrent access. One of the main drawbacks of locking is the need to resolve deadlocks, especially global deadlocks in distributed environments. Another disadvantage is that for some special application the probability of a concurrent access is very small and the locking overhead decreases performance.

A non-locking and therefore deadlock-free approach to isolation is optimistic concurrency control (OCC). It acts on the assumption that conflicts are rare, so that locking causes only unnecessary overhead. Transaction execution is divided into the phases read, validate and write. In the read phase transactions are executed without intervention of the OCC. They are validated a posteriori in the validation phase, where conflicts with other transactions are detected. Conflicts are resolved by rolling back an involved transaction. Successfully validated transactions reach the write phase and their changes are made persistent and visible to others.

Existing validation schemes could be subdivided into backward (BOCC) and forward (FOCC) oriented methods. BOCC takes only already finished transactions into account whereas FOCC validates against running ones. It is essential for both to be able to determine the read and write sets of transactions.

We integrated OCC into our object repository OPAS and therefore replaced the existing solution using two phase locking and pre-claiming. In the application domain of CASE systems conflicts are unlikely. The validation is done in a distributed manner. For this reason only the BOCC approach was applicable, because there is no efficient way to determine all currently running transactions. The validation authority is partitioned among all nodes and accommodated to the data distribution in order to maximize locality. An object is validated on the node where it is stored.

Local serializability is achieved as follows. The start time, end time and mode (read,written) of an object access are used for validation. A conflict is occurred if the given interval overlaps with another of an already successfully validated change transaction.

This test is our original extension of the BOCC validation algorithm. Traditional validation algorithms consider only object access in the associated read and write sets without the concrete point in time of the access. As a consequence, these algorithms mark certain transaction schedules as conflicting although the schedules are serializable.

The presented validation schema considers the concrete points in time of object access. Therefore it is insensitive to false conflicts whereas the test itself remains simple. A prerequisite for the validation schema is the existence of synchronized clocks within the cluster computer. This drawback is acceptable because the local system clocks of the the cluster nodes are regularly synchronized in order to run the Andrew File System (AFS).

But the given criterion for local serializability does not guarantee a sufficient level of isolation. The validation order of sub-transactions belonging to different global transactions could differ. To reach global serializability transaction numbers (TAN) were introduced. They are unique, linearly ordered and therefore comparable. For each object the TANs of the already validated transactions last read and last written are stored. A conflict with respect to global serializability has occurred if the read object was meanwhile written (own TAN less than the TAN of the object for last write access) or the written object was meanwhile read (own TAN less than the TAN of the object for last read access). This procedure produces the same validation order on all nodes and guarantees global serializability.

The validation is done on a basis of sub-transactions in the read phase. Thus there is no global validation phase. For this reason already validated changes are uncertain and therefore must be protected until the write phase is reached. This is done with locks on changed objects. We were able to integrate the validation into the two phase commit protocol to reduce communication. Compared to the former locking combined with pre-claiming the new approach offers a higher degree of parallelism. The achieved performance is presented in section 6.

5 Load Distribution Approach

Load distribution has been well-investigated in operating systems theory. In centralized realizations only a single instance makes decisions concerning load distribution, whereas a distributed realization lacks this instance. Load is distributed by initial placement and could be combined with migration schemes, the movement of an already placed load. The processing time of a parallelized request is determined by the processing time of the slowest sub-operation. Execution skew refers to the variance in processing time and often goes back to data skew. Therefore data allocation has a great impact on load distribution especially for shared-nothing systems. It determines how much data locality could be utilized during request execution.

In OPAS data allocation is done on a per-object basis without replication. A hash function assigns an object to a certain node. Furthermore, client requests are divided into sub-operations in order to utilize intra transaction parallelism. These sub-operations are initially placed using a static and centralized schema, and are assigned to the node holding the associated data for the purpose of maximizing locality. They are executed non-preemptively without migration.

With increasing number of processing nodes the hashed data allocation schema lead to data skew. Because of the tight coupling between allocation and sub-operation processing, execution skew arose, too.

First of all, we replaced the hashed allocation by a round robin strategy in order to achieve a broad object distribution and minimize data skew. Furthermore, we introduced a two level load sharing. On the first level, sub-operations were scheduled in a central and dynamic way. Initial placement decisions are made using a load index and capacity of the execution units. On the second level there is a randomized workstealing. It could be classified as a distributed load sharing approach. The basic idea is that underloaded nodes send steal messages to other randomly selected participants. Nodes which receive steal messages while they are overloaded send a certain number of sub-operations to the initiator. We present the results of the performance tests in section 6.

6 Achieved Results and Conclusion

The first implementation Parsytec-OPAS was designed for the parallel computer Parsytec GC/PowerPlus128 (shared-disk) [5], [6]. To get data for our experiments we considered structured analysis where the top-down method of stepwise refinement will be used. The application of the principle of decomposition creates hierarchical data structures where each set of neighbors represents a level of abstraction.

Using a sequential machine, object components will be searched after each other. When using a parallel machine object components which have the common father can be searched in parallel. This kind of parallelity will be called as the intra-object parallelity. In data processing typical for CASE tools there are frequently used operations like GET OBJECT and SHOW OBJECT. This means, that all components of asked aggregated objects have to be found on the disk, in the object buffer or in the page buffer [5] and displayed on the screen. We used data of a CASE tool collected during analysis of two projects. Their features are in [6]. The measured speed-up for intra-object parallelism as shown in Fig. 1 has already been published in [6]. We found that there is a important influence of the number of instructions that accompany the fetch of an object. These instructions represent the object processing between its fetch and its use. With growing number of such instructions the advantages of parallel processing are growing, too.

To compare performance of both new implementations with the old system performance we used the same data.

The system had been ported to the "Chemnitzer Linux Cluster (CLiC)", which has the following features: 528 nodes, each has 512 MB RAM and 20 GB local disk, 800 MHz frequency, fast Ethernet (100 MBits/s) node communication, throughput maximal 128 GBit/s, minimal latency 100 μs.

Parsytec-OPAS was adapted for the shared-nothing environment of CLiC using CORBA [2]. We refer to the system as CORBA-OPAS. It makes use of MICO as implementation of the CORBA standard. Another version of the object server (called MPI-OPAS) uses the Message Passing Interface (MPI) and the aspect weaver AspectC++. It is presented in [7].

The speed-ups for both implementations for intra-object parallelism are shown in Fig. 2 and 3. The performance of CORBA-OPAS is slightly better, although the graph shapes mainly correspond. Compared to Parsytec-OPAS both CLiC implementations perform better. The exchanged concurrency control mechanism provides an explanation

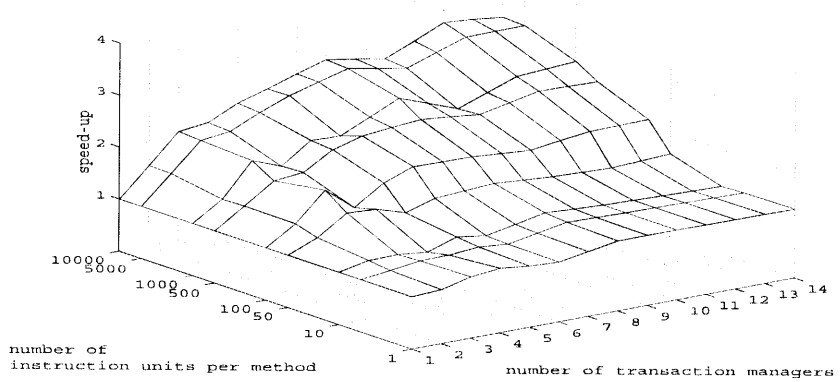

Fig. 1. Parsytec-OPAS speed-up for intra-object parallelism

for the arisen difference. The former implementation used locking combined with pre-claiming, whereas the latter benefits from optimistic concurrency control.

We compared the performance of the two concurrency control mechanisms (locking and OCC) as used in MPI-OPAS. The outcome is as follows. Optimistic concurrency control outperforms the locking in almost every measured point. For the series using fewer instruction units, the speed-up is about 20% higher on the average, whereas the maximum speed-up is increased up to 30%. Concerning the series with a greater amount of instruction units, the increase of the speed-up turned out to be smaller and levelled off at about 10% on the average. The results indicate that OCC not only increases the actual system performance, but also raises the inherent potential for parallelization.

Fig. 4 and 5 show the achieved results concerning inter-object parallelism for CORBA-OPAS and MPI-OPAS. Both implementations performed almost identically. Particularly for a greater amount of clients the speed-up increased nearly linearly. Due to optimistic concurrency control and load distribution, performance could be increased. In former versions like Parsytec-OPAS, but also CLiC-OPAS versions with pessimistic concurrency control, speed-up did not exceed 15. The dependency between processor nodes and speed-up was not linear, but logarithmic in these versions.

Although the speed-ups for both CLiC implementations are almost identical, the runtime differs greatly. Fig. 6 presents the runtime curve for the CORBA-OPAS. MPI-OPAS is about 15 – 25 times faster than CORBA-OPAS as shown in Fig. 7. It is already very well known that the usage of middleware like CORBA causes overhead, but we did not expect that the overhead ratio would be that high.

Furthermore, it is interesting that the impact of applied optimizations on the performance was different for the two versions. CORBA-OPAS did benefit greatly from load sharing. Its maximum speed-up could be increased by 41% from 22.4 to 31.6 (see Fig. 4). Initial placement of objects was not considered with respect to CORBA-OPAS and therefore had been left unchanged.

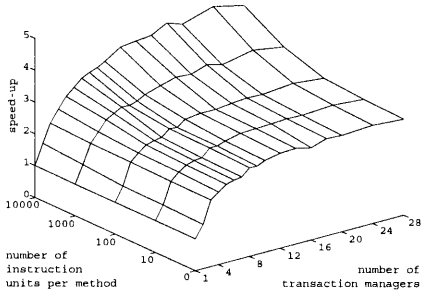

Fig. 2. CORBA-OPAS speed-up for intra-object parallelism

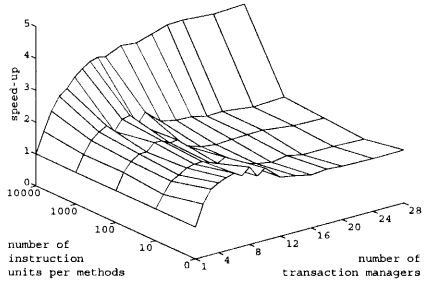

Fig. 3. MPI-OPAS speed-up for intra-object parallelism

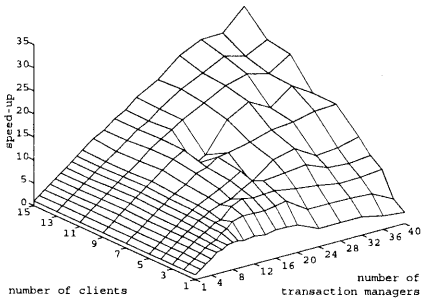

Fig. 4. CORBA-OPAS speed-up for inter-object parallelism

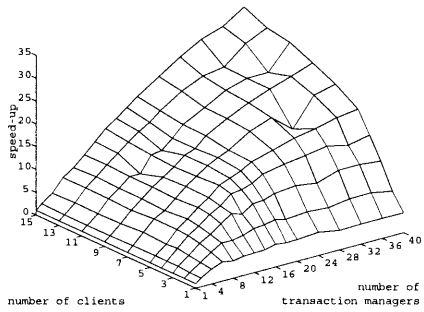

Fig. 5. MPI-OPAS speed-up for inter-object parallelism

The benefits from dynamic load sharing as mentioned in section 5 were disproportionally smaller for MPI-OPAS. In terms of numbers they are 3.8% on the average and 1.0% at the maximum. Whereas the initial placement of objects had a stronger effect on the performance, the round robin strategy reduced data skew and raised the speed-up by 32% from 25.4 to the maximum of 33.4 as displayed in Fig. 5.

For MPI-OPAS, we measured the impact of concurrency control on performance for inter-object parallelism. The result is that OCC outperformed pre-claiming locking. In direct comparison speed-up for OCC is about 60% higher. Thereby the ratio between the speed-ups did increase for a greater number of processing nodes. Consequently this behavior causes the almost linear speed-up rising illustrated in Fig. 5. Furthermore, we measured the influence of synchronization conflicts on performance. The speed-up decreased while increasing the conflict rate. Without conflicts, the maximum speed-up

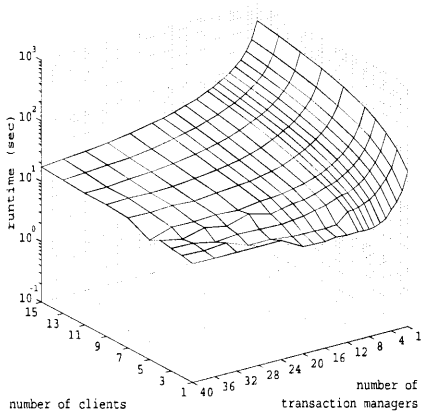

Fig. 6. CORBA-OPAS runtime for inter-object parallelism

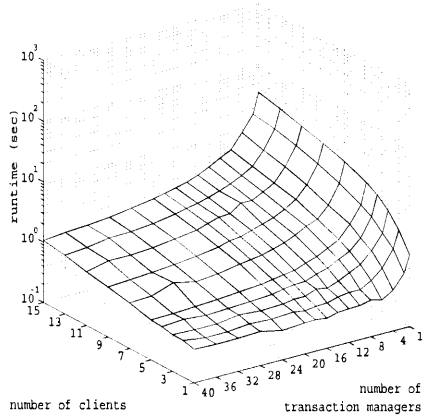

Fig. 7. MPI-OPAS runtime for inter-object parallelism

was 33.4. At a conflict rate of 1 % (resp. 5 %, 10 %, 20 %) the maximum speed-up decreased to 30.8 (resp. 29.0, 27.4, 25.5).

We believe that our results can be used for performance and speed-up estimation of proposed cluster systems and also for the choice of implementation technology.

References

1. Emmerich, W., Kroha, P., Schäfer, W.: Object-oriented Database Management Systems for Construction of CASE Environments. In: Marik,V., Lazansky,J.,Wagner,R.R.(Eds.): Proceedings of 4th International Conference DEXA'93, Lecture Notes in Computer Science, No. 720, Springer, September 1993.
2. Fleischer, M.: Tuning of CORBA-based Object-Repository in Shared-Nothing Cluster Environment. M.Sc. Thesis, TU Chemnitz, 2003 (In German).
3. Gruber, R. E.: Optimism vs. Locking: A Study of Concurrency Control for Client-Server Object-Oriented Databases. Technical report MIT/LCS/TR-708, 1997.
4. Kroha, P.: Objects and Databases. McGraw-Hill, 1996.
5. 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.
6. Kroha, P., Lindner, J.: Parallel Object Server as a Data Repository for CASE Tools. In: Croll, P. / El-Rewini, H. (Eds.): Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems PDSE99, pp. 148-156, IEEE Computer Society, ICSE99, Los Angeles, May 1999.
7. Kurth, M.: Tuning of Distributed Object-Repository OPAS. M.Sc. Thesis, TU Chemnitz, 2005 (In German).
8. Rahm, E.: Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems. ACM Trans. Database Syst., 18(2):333-377, 1993.
9. Rashid, A.: Aspect-Oriented Database Systems. Springer, Berlin, 2004.