Performance Evaluation of a Transactional DSM System

M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer and P. Schulthess
Department of Distributed Systems,
University of Ulm, Germany

Abstract

Distributed Shared Memory (DSM) is an interesting option to implement a distributed object system. Message passing, marshalling and remote invocation is replaced by the uniform DSM abstraction. The performance drawbacks of DSM Systems are often caused by complex distributed locking mechanisms. In response to this our multi purpose Plurix Operating System (OS) implements a transaction based DSM. Memory consistency is maintained by optimistic synchronization mechanisms and atomic transactions which have been used in database technology in the past. Such a transaction based DSM with optimistic synchronization guarantees a sequential consistent view on the shared data. After a short review of the Plurix DSM environment we present the communication protocol that implements transactional consistency. Furthermore we describe an experimental setup of a Plurix cluster which is used for the performance evaluation of the transactional DSM system Plurix.

Keywords: Distributed Shared Memory, Operating Systems, Memory Consistency, Experimental DSM-Setup, Measurements

1 Introduction

Commercial Operating Systems like Windows NT, Linux and MacOS offer communication services based on message passing combined with remote invocation. Traditional socket interfaces force developers to superpose their own network protocols, to handle a plethora of error conditions. This increases software complexity and the costs of maintenance. Higher level interfaces are offered by the Remote Procedure Call (RPC) and object oriented approaches like RMI, CORBA and .NET. These higher level APIs offer a rich set of functions but fail to simplify the design of distributed software systems. The development effort is typically spread across several software layers and development tools. The available middleware centers on providing distributed database functionality and messaging, but no consistency.

Moving the distribution functionality into the OS is an interesting alternative. This can be achieved by a Distributed Shared Memory (DSM) mechanism providing a virtual address space shared among tasks on loosely coupled processors, like introduced by Keedy [1] and Li [2]. The application programmer is offered a transparent view at shared data on several nodes connected via a network. Regular pointers are used for both local and remote memory accesses. OS and memory management hardware jointly will detect a remote memory access, fetch the desired memory block and maintain memory consistency.

In the Plurix OS we not only provide the distributed virtual memory abstraction, but each program inherits the potential to maintain consistent data structures in shared memory. Restartable transactions combined with optimistic synchronization facilitates the OS to restart operations and to guarantee a sequentially consistent view for each application on the shared data.

The overall system structure of Plurix is patterned after the innovative Oberon system de-

veloped by Wirth and Gutknecht [3]. Each station uses a central event loop and a simple cooperative multitasking concept. Plurix has been successfully demonstrated at various trade fairs and proved to be fast, compact and reliable.

Our paper consists of six sections. Section two gives an overview of existing page based DSM implementations including Plurix. Section three describes transactions and the optimistic synchronization scheme of the Plurix operating system. Section four presents an experimental setup and section five outlines preliminary measurement results. Finally we will give an outlook to future work.

2 Overview of existing page based DSM Systems

An early idea of DSM was presented by L. Keedy in 1985 [1]. Since then, the research interest in DSM Systems has grown steadily. A multitude of software or hardware based systems and hybrid architectures have been developed [4]. We do not attempt to give a comprehensive perspective of the state of DSM systems in this section. However, because Plurix is a page based system we shortly review some representative paged based systems: IVY [2], Mirage [5] and TreadMarks [6].

Page based DSM systems detect memory accesses to pages by using the protection features of the MMU. MMU hardware support can substantially speed up program execution in comparison to software based implementations but it is afflicted by the false sharing problem. The term "false sharing" is only defined vaguely in the literature [6]. False sharing is a characteristic performance penalty of page based DSM systems and occurs when two semantically independent variables reside on the same page and are alternatingly accessed by different processors. As a result the page is exchanged repeatedly between the processors. It is crucial to choose an appropriated page size. Large pages can speed up memory access due to the locality of data [7]. On the other side, false sharing increases when larger memory pages are used.

2.1 IVY

IVY [2] was one of the first proposed DSM systems. It is a userlevel implementation running on a group of network processors, the Apollo Domain system. It implements a page based DSM allowing multiple readers but only one writer per page. Memory consistency is guaranteed by an invalidation protocol, which requires that all readonly copies of a page are invalidated before a processors writes to a page. Sequential memory consistency is enforced in much the same fashion as in tightly coupled multiprocessors. This is the main reason why the performance of IVY is not convincing. Additional overhead is introduced by the user level approach.

2.2 Mirage

In contrast to IVY, Mirage is implemented in the kernel of an existing operating system [5]. The main idea is that a writer of a page should retain access to that page for a fixed period of time Δ. This can improve the exploitation of processor locality and avoid trashing. The value of Δ may be dynamically tuned. Mirage handles memory segments which are partitioned into pages. A process creates a segment by defining its size, name, and access protection. All other processes locate and access the segment by name. Requests are sent to the creator of the segment, where they are sequentially processed. The performance of the whole system is highly sensitive to the proper choice of the parameter Δ value.

2.3 TreadMarks

TreadMarks [6] is a userlevel implementation on top of readily available Unix systems. It applies the lazy release consistency model, together with a page invalidation protocol, that allows multiple concurrent writers to modify the page. On the first write to a shared page, a copy called "twin" is made. The "twin" can

later be compared to the current copy of the page in order to make a “diff” a record that contains all modifications to the page. Lazy release consistency does not require “diff” creation at each release (e.g. like Munin [8]), but allows it to be delayed. Lazy release consistency can achieve better performance than the release consistency implemented in Munin.

2.4 Architecture of Plurix

Plurix is a standalone PC operating system (min. 80486) written in Java implementing a proprietary Java compiler. Plurix runs within a single LAN segment. The central abstraction within our design is a global address space shared by several nodes. The global address space is organized as distributed heap storage (DHS) containing both data and code. Accesses (read or write) are properly monitored by the MMU. Tasks in the OS are partitioned into restartable transactions. Memory persistence and backup storage is provided by a special node, the Page Server, equipped with a large harddisk. The DHS in Plurix introduces a new model to maintain memory consistency: Restartable transactions and optimistic synchronization [9], [10] and [11]. This memory consistency model satisfies the requirement of the sequential consistency and of the *transactional consistency* as introduced in [12]. Plurix is designed as a multi purpose OS and high performance computing is currently not one of our main research goals.

3 Synchronization in Plurix

This chapter presents two variants of optimistic synchronization: forward and backward validation using restartable transactions. This allows concurrent access to shared objects in the Distributed Heap Storage from several nodes in a Plurix cluster, whereby collisions are resolved by restarting conflicting operations.

3.1 Restartable Transactions

Transactions must be restartabel in order to adhere to the well known ACID (Atomicity, Consistency, Isolation, Durability) paradigm [13]. In Plurix an aborted transaction and its input is automatically resumed at some later time.

Atomicity: A transaction groups several read and write operations to the DHS into an atomic indivisible operation. If for any reason a transaction cannot be completed, every modification of this transaction will be restored to the state before its beginning, by using shadow copies.

Consistency: Transactions always begin and terminate on a consistent view of memory. When a transaction starts, the DHS is in a consistent state and at the end, the transaction leaves the DHS again in a consistent state.

Isolation: Modifications are isolated, while a transaction is running. Therefore intermediate results are invisible to other transactions.

Durability: If a transaction commits all modifications persist (durability) until a successor transaction modifies the values.

These characteristics of transactions allows the optimistic concurrency control to restart operations in case of a conflict between concurrent operations

3.2 Optimistic Synchronization

The basic expectation behind optimistic concurrency control is that most transactions do not conflict with each other. Therefore they can tentatively proceed without locks and conflicts are checked at the end of a transaction. If two or more transactions collide, at least one needs to be aborted and may be restarted later. If they do not conflict all transactions can commit successfully.

Collision detection can be done as a forward or as a backward validation.

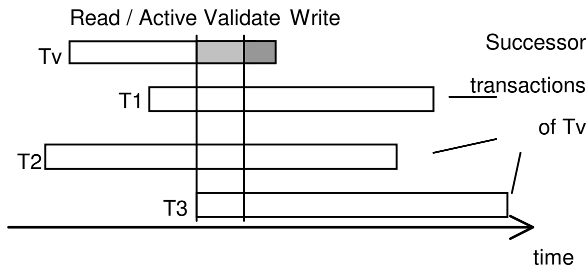

Figure 1: Forward Validation

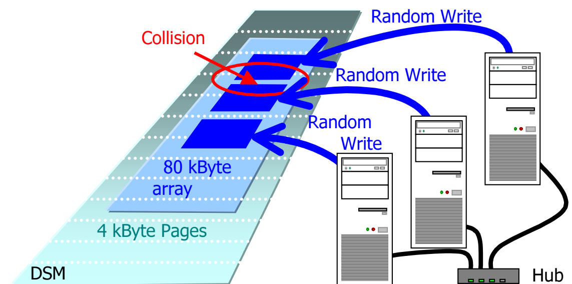

Figure 2: The test setup

- In a backward validation scheme the addresses of all modified objects (write sets) of a transaction are compared against all overlapping transactions that already have committed. If a conflict is detected, the currently validating transaction is aborted; if there is no conflict the transaction can commit.
- If forward validation is used the write set of a validating transaction is compared against all active transactions. If a conflict is determined one or more transactions must be restarted.

The forward validation scheme has the advantage that there are more than one candidate transactions to abort in case of a collision. Therefore a fairness protocol can help to decide which transaction to abort.

3.3 Collision detection

During the validation phase at the end of transaction Tv, collisions between concurrent transactions are detected by comparing the read references and the write references of participating objects. Three rules are to be satisfied to exclude conflicting read or write operations [14].

1. Read-Write: Tv must not read an object written by an overlapping Transaction Tj.
2. Write-Read: Tj must not read an object written by an overlapping Tv.
3. Write-Write: Tj must not write an object written by Tv and vice versa.

Tv is a transaction in the validation phase and Tj are running transactions overlapping with Tv in the case of forward validation. For the synchronization in Plurix one commit-token in the DSM is used to ensure that only one node in the cluster commits a transaction at the same time.

4 Experimental Setup

The measurements objective is to study the performance of the DSM communication protocols during the operation of a Plurix cluster. The page-fault mechanism and the communication for the synchronization of the cluster is investigated in particular.

A test application is introduced in this chapter which allows several computers to access and modify elements concurrently of a shared array. This scenario was chosen because it allows to interpret the measured data in spite of the dynamic behaviour of the protocol and the distributed memory accesses.

4.1 The Plurix test cluster

The cluster setup to perform the measurements consists of eight PCs between 233 MHz and 1GHz connected by a 100 Mbit/s Fast-Ethernet(Figure:2). Additionally, a long-integer array has been provided containing 10.000 elements within the DSM, published in the name service. The eight computers read and write on the shared array with the probability of collisions. On each of the computers two transactions are ac-

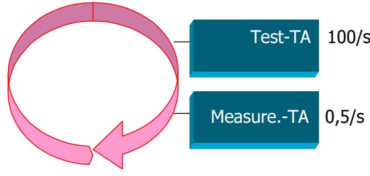

Figure 3: Transactions in the Plurix loop

tive: The test-transaction modifies elements in the common array every 10 ms and the measurement-transaction evaluates the protocol performance, every 2s(Figure:3).

4.2 The Test-Transaction

The Test-TA is started within every 10ms and modifies elements within the shared array. This is done as follows:

1. During the initialization phase, the transaction searches for the shared array in the cluster wide name service. In case of absence, an array of long-integers with 10.000 entries is created and registered in the name service.

2. Using a random value, an element within the array is selected and starting from that, the next 501 elements will be modified. If the end of the array is reached the transaction continues from the beginning of the array.

The size of the shared array and the number of elements described are variable within the application. However access of more than 500 elements is preferred, because in this case more than one page of the DSM is modified by every Test-TA.

The random starting point within the array has the advantage that concurrent access does not provoke collisions in each and every transaction, because this would contradict the optimistic hypothesis of conflicts being rare.

This application is started on all computers of the cluster and the measured values are newly calculated after every added computer.

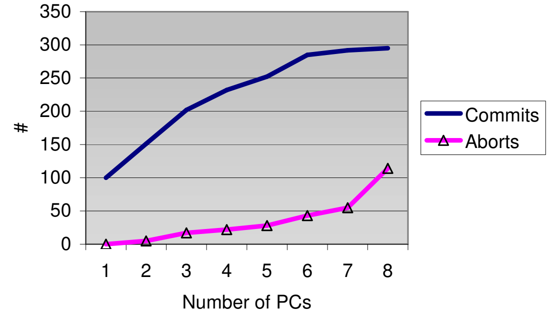

Figure 4: Number of commits and aborts

4.3 The measurement-transaction

The following values are determined in a two seconds interval by the measurement transaction: The average number of commits and aborts per second in the DSM and the average page request latency as well as the token request latency.

5 Results of the measurement

Figure 4 shows the number of commits and aborts with increasing numbers of computers. As long as the first computer operates alone in the DSM, it is able to commit successfully 100 transactions per second and therefore no abort and restart of transactions occurs. As soon as the second computer is added the number of aborts increases moderately, but the number of commits in the cluster is increased by 50%. The non-linear increase of commits is caused by two circumstances:

1. To carry out a commit each computer needs the common token. The possibility that the token of the last commit on a computer still exists is decreasing according to the increasing number of computers in the DSM and a transfer of the token becomes necessary.

2. By the writing access to the common array the modified pages are invalidated on all other PCs. That causes aborts in case of concurrent write to the pages, but in case

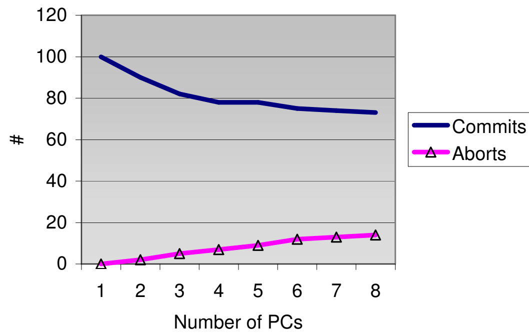

Figure 5: Number of commits and aborts of the first PC

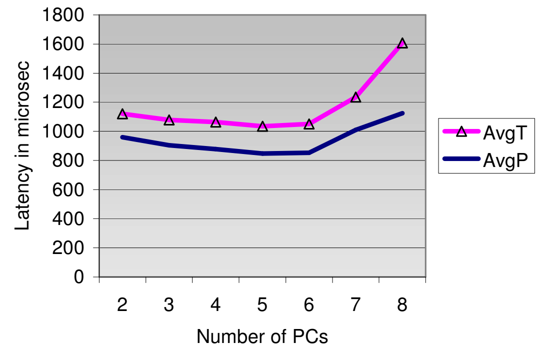

Figure 6: Average page and token latency

of no collision the pages are invalidated anyway. At the next access the pages are again requested from the DSM.

This effect increases if more computers are added to the DSM. The shared array occupies 21 pages of the DSM. Concerning eight computers whereby each of them modifies two pages on radomly determined positions, that leads to 16 page accesses of the Test-TAs on a shared object consisting of 21 DSM pages. This does not force collisions for every page access but invalidates most of the copies of the 21 pages in the DSM during one operation cycle of the Test-TAs.

Figure 5 shows the condition of the first computer concerning the number of commits and aborts while the test application is started by other computers. The first computer contains a 1GHz processor. It shows that the other computers are influencing the fast one moderately only. In the presence of eight active computers the fast computer still reaches 73 commits and only 14 aborts per second. In contrast to the 1GHz PC the eighth computer with 233MHz attains 3 commits only and 54 aborts per second. This is based on short execution periods of the fast computer. Because the computer modifies the 501 elements very fast and it retains yet the token of the last commit with great probability. The slower computer often sets the token free before ending his own transaction. Therefore the possibility of abort increases considerably.

Figure 6 shows the average latency during request of a page or a token. In this test as well the test-transaction has been started step by step on all computers and the average latency periods concerning all computers have been averaged:

- The latency for a token request is longer than the latency of a page request although the data to transfer during a token request is 64 Bytes compared to the 4kBytes of a page. This is because pages are transferred any time on demand and the token is withheld until the last commit is complete.

- The protocols and the network hardware drivers need to be optimized. They could not yet reach the upper physical bandwidth of the Fast-Ethernet network. The page-fault, consisting of a page request message and the transfer of the page data, needs more the $800\mu s$ whereas regarding the network this should take around $450\mu s$.

- The token and page latency increases for seven and eight PCs. This is caused by the number of token and page requests which slows down the response time in the cluster.

6 Conclusions

Early performance evaluation has shown that the Plurix protocols achieve the requirements of a page-based DSM with optimistic concurrency control, whereas optimizations of the hardware drivers and the protocol implementation should be realized.

The specified and implemented communication protocol allows to develop applications producing a large number of collisions in the DSM, leading to a high system load. Not only the presented test but also the use of the protocols during trade fairs (CeBIT2000, CeBIT2001) for several days has demonstrated the stable function of the DSM protocol.

Future work should be done on the false sharing problem of a page based DSM. Currently a tool for the detection of false sharing is under construction. Furthermore a fairness protocol in the DSM shoud be developed, to protect slow PCs from starvation in case of excessive collisions.

References

[1] J.L. Keedy and D.A. Abramson. Implementing a large virtual memory in a Distributed Computing System. In *Proc. of the Hawaii International Conference on System Sciences*, Hawaii, USA, 1985.

[2] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proceedings of the International Conference on Parallel Processing*, 1988.

[3] N. Wirth and J. Guteknecht. *Project Oberon - The Design of an Operating System and Compiler*. Addison-Wesley, 1992.

[4] Department of Computing Science. A comprehensive bibliography of distributed shared memory. Technical Report TR9617, University of Alberta, 1996.

[5] B. D. Fleisch and G. J. Popek. Mirage: A Coherent Distributed Shared Memory Design. In *Proc. 14th ACM Symp. Operating Systems Principles*, 1989.

[6] P. Keleher et al. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *USENIX Winter 1994*, 1994.

[7] A. Cox W. Zwaenepoel K. Rajamani C. Amza. Tradeoffs Between False Sharing and Aggregation in Software Distributed Shared Memory. In *Principles and Practice of Parallel Programming*, 1997.

[8] J. K. Bennet W. Zwaenepoel, J. B. Carter. Implementation and Performance of Munin. In *13th ACM Symposium Operating System Principles*, 1991.

[9] J. T. Robinson H. T. Kung. On Optimistic Methods for Concurrency Control. In *ACM Transactions on Database Systems*, 1981.

[10] S. Traub. *Speicherverwaltung und Kollisionsbehandlung in transaktionsbasierten verteilten Betriebssystemen*. PhD thesis, University of Ulm, Germany, Distributed Systems Department, 1996.

[11] M. Schoettner S. Traub and P. Schulthess. A transactional DSM Operating System in Java. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 1998.

[12] M. Schoettner M. Wende and P. Schulthess. Optimistic Synchronization and Transactional Consistency. In *The Fourth International Workshop on Software Distributed Shared Memory WS-DSM'02*, Berlin, Germany, 2002.

[13] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme Grundlagen, Konzepte, Realsierungsformen*. Springer-Verlag, 1996.

[14] G. Coulouris, J. Dollimore, T. Kindberg. *Distributed Systems. Concept and Design*. Addison-Wesley, 2001.