Architectural Issues in Adopting Distributed Shared Memory for Distributed Object Management Systems*

Jung-Ho Ahn†, Kang-Woo Lee‡, and Hyoung-Joo Kim†

†Department of Computer Engineering
‡Department of Computer Science
Seoul National University
Seoul, KOREA 151-742

Abstract

Distributed shared memory (DSM) provides transparent network interface based on the memory abstraction. Furthermore, DSM gives us the ease of programming and portability. Also the advantages offered by DSM include low network overhead, with no explicit operating system intervention to move data over network. With the advent of high-bandwidth networks and wide addressing, adopting DSM for distributed systems seems to be attractive. In this paper, we propose two alternative distributed system architectures which are attempts at adopting DSM for distributed object management systems. The two proposed architectures are distributed shared cache (DSC) architecture and distributed shared recoverable virtual memory (DSRVM) architecture. We address several major issues in the proposed architectures.

1 Introduction

The growth of computer hardware and software technology extends database application areas, such as CAD/CAM, knowledge base, office information systems, and engineering applications. In these new applications, object management systems become essential. The rapidly increasing demands of managing distributed artifacts, together with the growth in the object technologies, have brought us to the development of distributed object management systems.

In general, the popular programming paradigm of current distributed systems is the message-passing paradigm. Message passing interface, however, forces the programmer to use different paradigms than shared memory interfaces used in a single site. Therefore, it is well known that the programming in distributed systems difficult to perform and inefficient[16][2].

Distributed shared memory (DSM) provides transparent network interface based on the memory abstraction. DSM, which has been an active area of research since the early 1980s, gives us the ease of programming and portability. Also the advantages offered by DSM include low network overhead, with no explicit operating system intervention to move data over networks. But, so far, only a few experience with applications using DSM exist.

Nowadays, with the advent of high-bandwidth networks and wide addressing, adopting DSM for distributed systems seems to be attractive.

In this paper, we propose two alternative distributed system architectures which are attempts at adopting DSM for distributed object management system and address some of the major issues in the proposed architectures, which are distributed shared cache (DSC) architecture and distributed shared recoverable virtual memory (DSRVM) architecture.

2 Distributed Shared Cache Architecture

Contemporary relational database systems support client-server environment which is typically based on query-shipping architecture where the server part processes queries which are shipped from clients. In this approach, page cache in a server machine should be shared so that multiple instances of the database server run in parallel. In contrast to traditional database systems, object-oriented database systems usually ship data (namely data-shipping) from servers to clients so that clients can navigate data and perform query processing by itself[5]. Although this type of client-server architecture is emerging as a popular paradigm to support data sharing over computer net-

*This work was partially supported by Ministry of Education through Inter-University Semiconductor Research Center (ISRC 94-E-2032) in Seoul National University.

0-8186-7125-4/95 $04.00 © 1995 IEEE

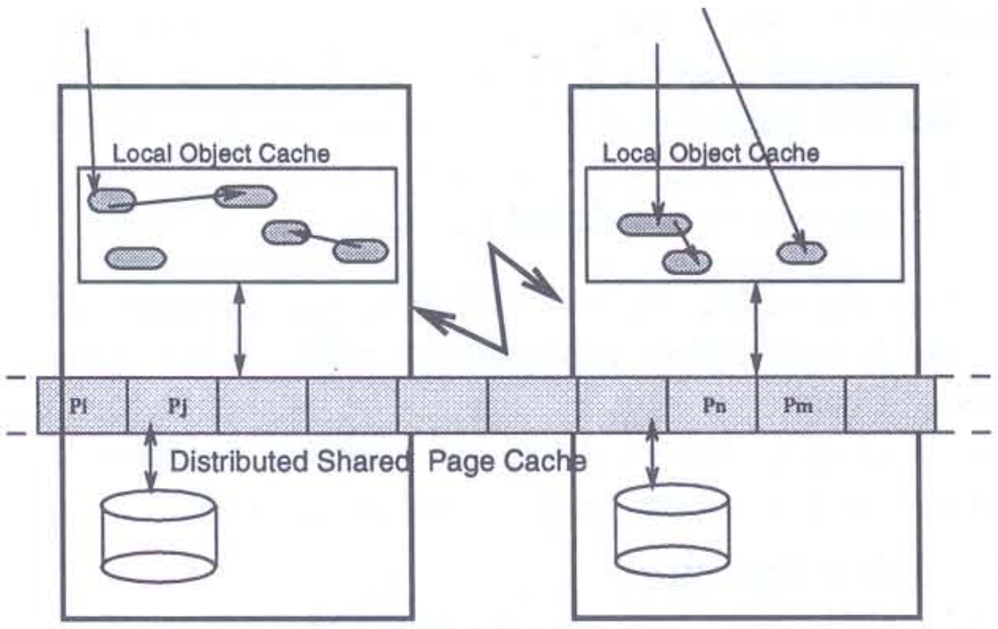

Figure 1: Distributed Shared Page Cache Architecture

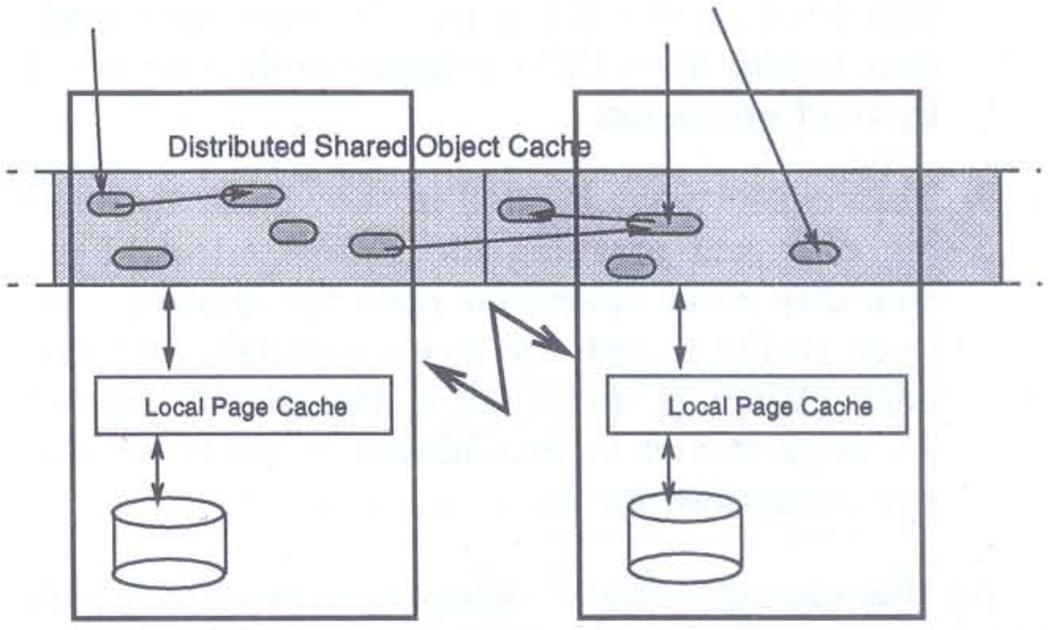

Figure 2: Distributed Shared Object Cache Architecture

works, some problems like concurrency control and cache consistency make the implementation of object management systems (OMSs) difficult.

We take the view that sharing cache over distributed shared memory may lessen the difficulty of the development of OMSs, since underlying DSM system takes all responsibility of maintaining cache consistency. And yet, because data manipulation requirements of object management systems are quite different from those of traditional systems, many existing systems maintain a separate object cache in addition to a conventional page cache[5].

Along this line, three architectures can be considered: *distributed shared page cache (DSPC)* where distributed OMSs share data in the granularity of pages and *distributed shared object cache (DSOC)* where systems share individual objects. Figure 1 and 2 show the corresponding architectures. Also both caches may be shared as the third.

There are tradeoffs in the use of DSM as an object cache relative to DSM as a page cache. In DSPC architecture, the granularity of a DSM is usually the same with the size of the unit of page caching – a multiple of hardware page size. So, there is no false sharing and smaller ping-pong effect(thrashing) than DSOC architecture. But because this architecture doesn't share an object cache, there must be a protocol among OMSs to ensure that the object caches of each site remain consistent with the shared database[18][4]. Also, object fault and swizzling[19][15] must be handled on each site.

On the other hand, object cache coherence problem does not occur in DSOC architecture. Underlying DSM maintains the consistency of object cache automatically. This makes it easy to support inter-transaction caching. Object fault handling and swizzling overhead can be diluted with sharing it, since fault handling and swizzling are needed to be done only once for each object.[1] DSM system fetches a remote memory in the unit of memory coherence. It means that DSOC architecture transfers a group of objects rather than one object at a time. This may lead to less communication overhead than pure object server architecture due to clustering objects. But DSOC architecture has a few disadvantages. One of them is that DSOC may cause more false sharing, where two sites compete for access to different data items in a single DSM coherence unit. The other is that it is difficult to implement than DSPC architecture, since object cache sharing is a difficult problem naturally.

However, it has a potential poor performance problem to simply extent a single-site multiprocess architecture to distributed one by adopting pure DSM. In the following two subsections, we address two major issues that might otherwise lead to poor performance and give some feasible solutions to them.

2.1 Cache Replacement

The cost of maintaining the strict memory coherence of DSM is likely to cause the overhead of the cache replacement. Therefore, special care must be exercised for cache management in DSC architecture. For example, when a site A selects a victim v kept in another site B by its own replacement algorithm, the cache replacement sequence is as follows:[2][6]

- If the page selected for replacement has been modified, it has to be written back to disk before the new page is read into it. However because the page is owned by a site B, flushing the page requires a remote paging from a site B to A. At

[1] All of the object caches should be mapped at the same virtual address to exploit these advantages

[2] We concern only about page cache whose page frame size is the same with the size of DSM memory coherence unit.

this point, a site A has got the page with read-only mode, since DSM memory fault is occurred by read operation.

- The request page is fixed in the victim by reading disk and marking its cache control block. But disk read operation calls for another network traffic to get the write ownership for the page. What is worse, all of the shared copy of the page should be invalidated to get the exclusive ownership for it.

As this example shows, selecting a remote victim requires higher network overhead than local page replacement. Now, we suggest two replacement strategies exploiting with tight cooperation between DSM and the cache management module of OMS.[3]

The first one is to use a new explicit remote paging interface. The way to make a shared memory segment accessible to an execution site can be implicit and explicit. Implicit method is based on page fault of operating system. The explicit one uses DSM interface directly. Implicit remote paging requires the cost of handling page-fault by virtual memory of operating system. But explicit remote paging may avoid the cost and also can give useful hints easily.

Because a victim will be overwritten as soon as it is shipped from its previous owner, transferring the page is not necessary. Thus, a new explicit interface which does not make unnecessary remote paging can be added for reducing the performance degradation. Addition to '`get`' primitive for getting a page in the specified mode from its owner, we propose a new interface '`get_new`'. It gets a page in the exclusive write mode without transferring page itself. Figure 3 shows how the '`get_new`' operation works.

The second one, we propose, is *replacement-cost hints* algorithm. This replacement strategy partitions cache space by their ownership and gives priorities to each of them according to their replacement costs. Replacement cost of each set shows how costly to replace a victim on a distributed shared memory space.[4]

The basic idea underling *replacement-cost hints* is the following:

1. Pages are organized into *victim sets*, where a victim set consists of all of pages which have a same replacement cost. Victim sets are arranged by their replacement-cost order.

2. Cache searches its victim sets in inverse order, starting from the lowest replacement cost victim set.

3. The first found favored page is selected as victim. A favored page in a victim set i is a page which has not been accessed for a certain time ϵ_i, where $\epsilon_i \leq \epsilon_{i+1}$, $1 \leq i \leq n-1$, and n is the number of victim sets and ordered by replacement costs.

In this algorithm, it is trivial that any remote victim, which has higher replacement cost, is not gone out of the cache as long as there is any other favored page. And also, threshold ϵ_i for each set i prevents the pages in a working set from being replaced.

2.2 False Sharing

If the memory coherence unit of DSM is larger than transactional unit of OMS, it is likely that more than one site will write access to a single coherence unit. This is called *false sharing* and may induce thrashing, where a memory unit moves back and forth at such a high rate that any work cannot be done[16]. The granularity of DSM sharing should be same or smaller than the granularity of locking to avoid this kind of false sharing. But otherwise, mechanisms to reduce thrashing are require to assure reasonable performance of systems.

Two existing DSM systems[8][3] give solutions to this problem at the DSM level. Mirage system[8] guarantees that a reader or a writer possesses the sharing unit without interrupt for a specific time window Δ. This prevents the sharing unit from being stolen away before any work can be done. Although optimally tuned value for Δ may give high throughput decreasing network traffic, it is difficult to choose an appropriate value for Δ dynamically. Munin system[3] employs another solution to reduce thrashing. It uses different coherence protocols for each shared data type. Type information specified by a programmer may improve overall system performance. But it imposes a heavy burden on a programmer to predict the type of every shared data.

It is well known that sequential consistency is too restrictive and weakening the coherence requirement makes adopting DSM more viable. We can also get performance gain by combining two separate synchronization activities – memory coherence control of DSM and concurrency control of OMS[14].

While above two systems do not fully use the application specific knowledge in maintaining coherence of DSM, our new loose coherence protocol exploits synchronization activities of transaction manager. This method grounds on that the relaxed coherence semantic will allow more efficient shared accesses and concurrency control will synchronize access to shared data. It is described briefly as follows:

1. Addition to `READ` and `WRITE` modes, there are `SHARED-READ` and `SHARED-WRITE` modes in the

[3] In this work, we assume that all site are sharing disks

[4] When using a non-shared disk architecture, it includes the cost of flushing out

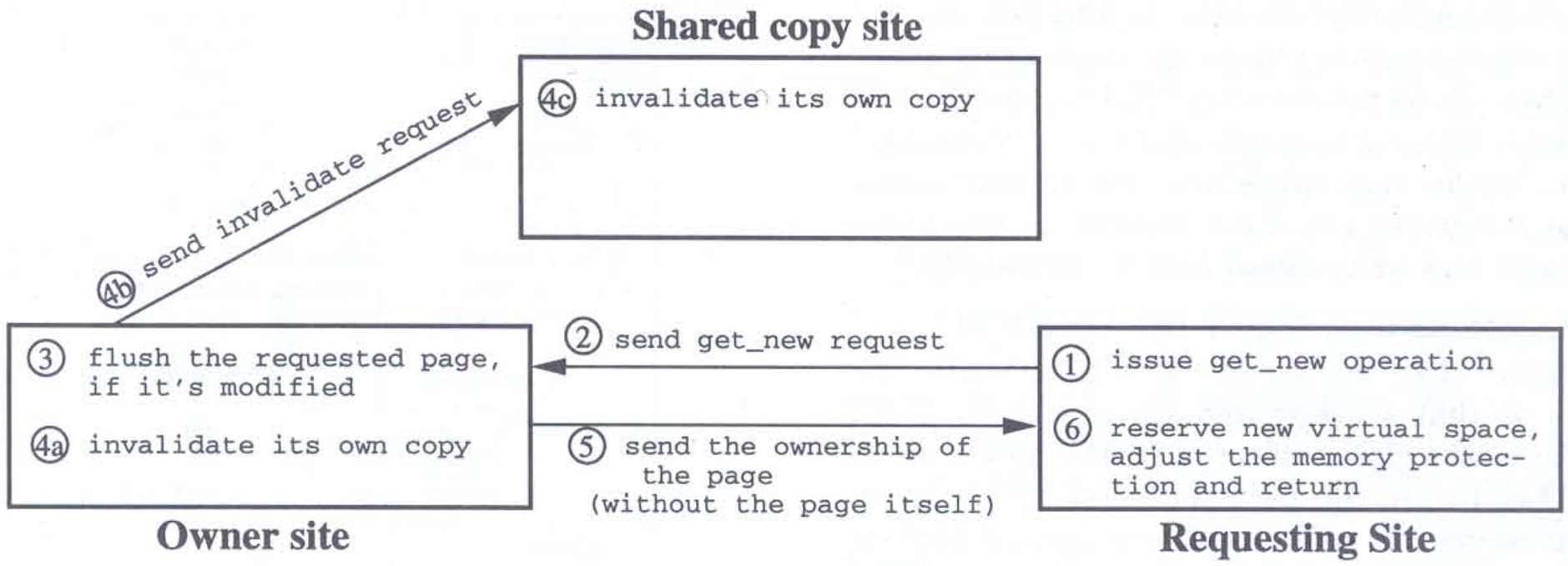

Figure 3: get_new protocol

distributed shared memory space. As the strict coherence protocol, multiple readers may exist at any instant for a single memory coherence unit, but only one writer may exist at any instant. Unlike these two basic modes, there can exist many shared-readers or shared-writers for a unit at the same time.

2. Initially, strict coherence protocol is used for each coherence unit. But if thrashing is likely to occur,[5] coherence protocol is weakened to reduce thrashing. There are two cases where the protocol is loosed.[6]

   (a) WRITE – WRITE : When an exclusive owner receives 'get_with_write_mode' request for a unit, the owner looses WRITE mode to SHARED-WRITE and returns SHARED-WRITE ownership to the request site instead of exclusive one.[7]

   (b) WRITE – READ : As the case 1, the owner changes the mode to SHARED-WRITER and returns SHARED-READ capability to the request site.

3. Once the protocol is weakened, get_with_write_mode or get_with_read_mode requests are handled by returning SHARED-READ mode or SHARED-WRITE for each.

4. When a transaction ends, all of the SHARED-READ or SHARED-WRITE units should be invalidated. If a unit is owned with SHARED-WRITE mode, it should be merged with other copies before invalidation. Data merging can be done by DSM server or by other shared owners distributively. It is worth to note that modified portion of data can be identified easily by recovery scheme.

In this protocol, because SHARED-READ or SHARED-WRITE mode does not incur any conflict for a shared unit, each site can service all conflicting accesses to it, but consistency is preserved by concurrency control mechanism. Thus, on the assumption that all shared data accesses should be done in well-formed concurrency control protocol, the suggested protocol works correctly.

3 Distributed Shared Recoverable Virtual Memory Architecture

Many researchers have studied issues in using operating system's virtual memory as caches in object management systems[7][17]. In this approach, databases are directly mapped into virtual memory and OMSs use persistent objects as transient memory objects. DSRVM approach is a sort of natural extension of this approach — mapping databases into DSM[12][13].

In DSRVM architecture, OMS never worries about where objects are and how to access them because object accessibility is governed by DSM system. Also, it does not have to implement distributed caching management and concern some aspect of cache coherency problem. As such, this architecture fully utilizes the advantages of DSM. Moreover, since most typical OMS applications show tighter working set than traditional ones, only little performance degradation is expected from using less DBMS-optimized caching.

As data are mapped into DSM and are directly manipulated in DSM, underlying DSM must support re-

[5] Thrashing can be detected by monitoring the transfer rates for each unit.

[6] This work excludes 'READ – WRITE' case for simplicity. But it can be done easily as other cases

[7] Data is also transfered.

coverable manipulations of data. This means, any anticipated crashes cannot violate the consistency of the data in DSM. To be recoverable, DSM system must be incorporated with log manager and recovery manager in OMSs. In the next subsection, we propose a protocol that integrates the cache coherence, two phase lock protocol and write-ahead logging protocol[10].

3.1 Transactional DSM for DSRVM

we assume that the system is a client-server architecture so that a designated server process (*DSM server*) knows the global status of DSM pages[8] over all nodes. Also, to provide permanence of DSM, we assume that server has non-volatile storages for backing DSM memory pages and it logs the changes in DSM page.

A client is a node participated in DSM complex and it is composed of *application transactions (APT)* and an *agent transaction (AT)* (see figure 4). An application transaction is an application process which is enclosed in transaction boundary. DSM system provides access control of DSM pages for their APT. When an APT tries to read(write) a DSM page which is not permitted to read(write), a page-fault is trapped. By this mechanism, DSM manager provides a transparent way to guarantee cache coherence, serializability, atomicity, and permanence. AT is a stand-alone process and it is only a transaction which interacts with DSM server so that it receives(sends) valid copies of DSM pages from(to) a DSM server on behalf of APTs.

According to the lock mode of DSM page, the status of a page is determined. When AT holds a lock in exclusive (EX) mode for a page, it implies the node has the invalid page. When AT has a lock in shared (SH) mode, the node has a valid copy of the page but no APTs in the client can write the page. When AT locks a page in NL mode (i.e. AT does not hold a lock for the page), the node has a writable valid copy of the page.

3.2 Client Protocol

The following is the scenario of the integrated protocol executed in client node.

- On client boot up: AT locks all pages of DSM in EX mode, which implies that all pages are invalid.
- On APT start: DSM manager of this process disables the access of all DSM pages so that any access to the DSM pages traps page fault.
- On read page P fault: Fault handler (FH) tries to lock the page conditionally in SH mode. If the request is not granted immediately and the lock holder is AT, FH sends `get_with_read_mode` message to AT and locks the page in SH mode again. After the lock is granted, FH marks the page readable and resumes the process.
- On write page P fault: FH tries to lock the page conditionally in EX mode. If the request is not granted immediately and AT is the one of the lock holders. FH sends `get_with_write_mode` message to AT and locks the page in EX mode again. After the lock granted, FH marks the page writable (which also means the page is readable) and resumes the process.
- On APT commit: If APT has any writable DSM pages, it requests AT to send `commit` message to DSM server with modified pages. After AT completes to send those pages, APT releases the all locks it holds and disables access to all pages.
- On APT abort: If APT has any writable DSM page, it requests AT to send `discard` messages for those pages. After AT completes to send messages, APT transfers the all exclusive locks it holds to AT and disables the access to all DSM pages. By *transferring exclusive locks to AT*, invalid access from any other lock waiting APT can be avoided.
- When AT receives `get_with_read_mode` (`get_with_write_mode`) message from APT: AT

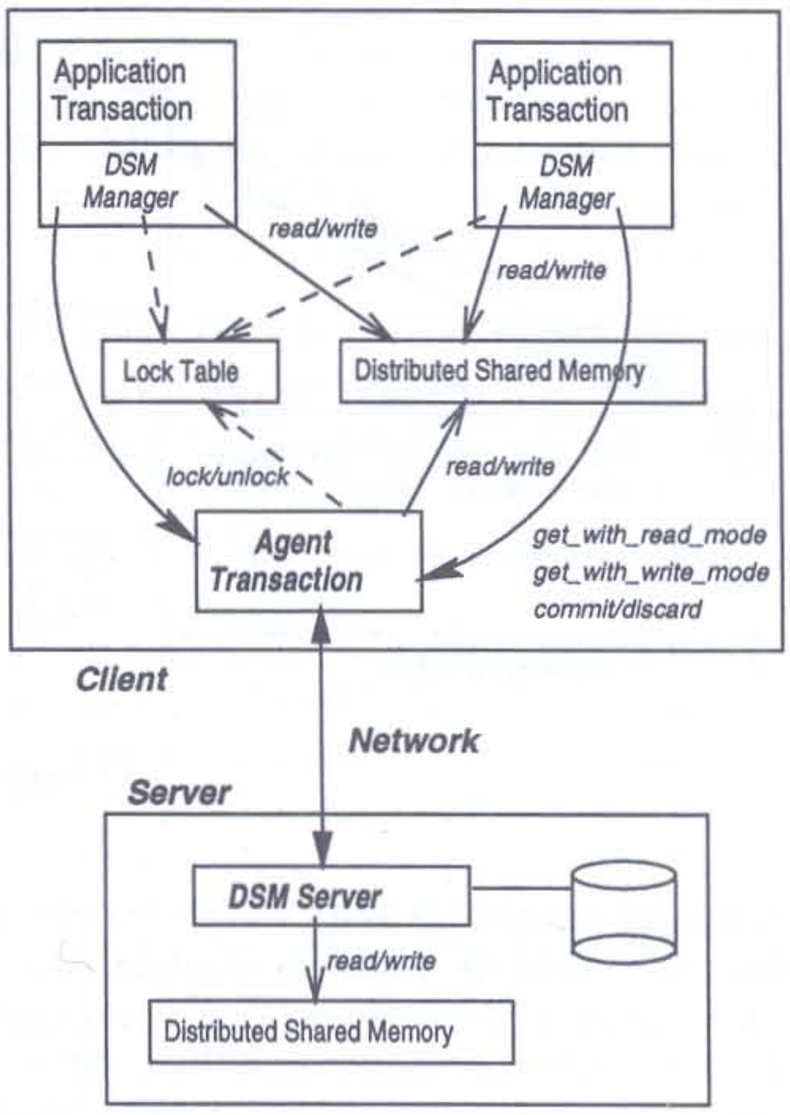

Figure 4: Architecture for Integrated Protocol

[8]It means a memory coherence unit of DSM.

forwards it to the DSM server. After received the valid copy (acknowledgment) from DSM server, AT downgrades its lock to SH (NL) mode.

- When AT receives `recall` (`invalidate`) message from the DSM server: AT tries to lock the corresponding page in SH (EX) mode. After the lock granted, AT sends the page (acknowledgment) to DSM server.

3.3 DSM Server Protocol

The following is the scenario of the server part of the protocol.

- On receiving `get_with_read_mode` message from a client C: DSM server checks to see DSM server has a valid copy of it. If server has, it sends the copy to the client C (more precisely, to the AT in the client C). Otherwise, it sends `recall` message to the page owner (say C'), and wait for valid page arrival. After receiving the valid copy of it, server marks C' as a page holder (instead of page owner) and marks also the client C as a page holder.

- On receiving `get_with_write_mode` message from a client C:

  1. If there is a page owner node for the page, it sends `recall` message to that client. After receiving the valid copy of it, server marks that client as a page holder.
  2. If there are any page holder nodes for the page, it sends `invalidate` messages to those clients. After receiving acknowledgments from all of them, server marks each clients as 'invalid-page holder'.
  3. Server sends a valid copy of the page to the requesting client C and marks C as the page owner.

- On receiving `commit` message from a client C: server receives valid pages from client and saves them into non-volatile storage of those pages.

- On receiving `discard` message from a client C: server marks the client C as 'invalid-page holder' for that page, and marks itself as a page owner.

Due to space constraints, we omit the correctness arguments of this protocol. But, since locks hold by a APT are released only after it commits(or abort), this protocol is two phase locking protocol. Also, the most recent page will be accessed by 'recall' mechanism, this protocol guarantees sequential memory consistency.

3.4 Pros and Cons of the Protocol

We believe this protocol has the following advantages. First, it integrates cache coherence protocol and locking protocol, so that the number of messages between DSM server and client can be reduced. When the separated protocol is used, almost two times more messages are required than the integrated one for APT to access a page. Second, this protocol takes advantages of data caching and lock caching, which may reduce the number of messages between clients and a server to access a page. An application transaction can read/write a DSM page without any server interactions, if that page is already cached in the client. Third, this protocol requires any specific interfaces for DSM manager except transaction commitment and transaction abort. An application transaction programmer does not have to lock pages, nor have to generate log records for DSM page updates. Fourth, this protocol does not require the complex two phased commit protocol, which makes transactional protocol complicated in most existing distributed DBMS.

But, this protocol has a few shortcomings. It supports only sequential memory consistency which may be too restrictive for some applications. But we worry that any relaxed memory consistency will result in database inconsistency in most OMS applications. And sometimes transaction concept makes a relaxed consistency protocol meaningless because locking protocol, in general, requires very restrictive memory consistency. This protocol supports only *FORCE* buffer strategy[11], which reduces transaction throughput. We understand that most commercial DBMSs having data shipping architecture use *FORCE* buffer strategy. Lastly, If some DSM pages are frequently accessed from several nodes, overall system is likely to fall into thrashing. We are currently studying to overcome this problem.

4 Related Works

Most of researches on DSM are emphasized on memory coherence, granularity of sharing, heterogeneity, avoiding thrashing and so on. Only a few works (Hsu and Tam[14], Hasting[12]) are done in implementing DBMS or atomic transaction using DSM.

Hsu and Tam's work puts an emphasis on performance enhancement by integrating cache coherence (coherent memory) and concurrency control (process synchronization). They show the performance enhancement using simulation study of two synchronization algorithms: *2PL-MC*, which separates transaction synchronization from memory coherence, and *2PL** which bypasses memory coherence. Based on simulation results, they argue that significant performance gain can potentially result from bypassing

memory coherence and supporting process synchronization directly on DSM. Hastings's work was to propose *transactional distributed shared memory (TDSM)* using Camelot[7] transaction facility, which provides recoverable virtual memory and *Mach external memory manager (XMM)*[9].

5 Conclusion

In this paper, we proposed two alternative distributed system architectures which are attempts at adopting DSM for distributed object management system: distributed shared cache (DSC) architecture and distributed shared recoverable virtual memory (DSRVM) architecture and addressed some of the major issues.

In DSC architecture, we explored the tradeoffs in the use of DSM as an object cache relative to DSM as a page cache. We also suggested a new replacement strategy exploiting the knowledge of the ownership of data items and provide some feasible solutions to false sharing problem.

The major advantage of DSRVM architecture is to provide transactional facilities for direct manipulations of data in DSM. We presented a new protocol for DSM to support transaction concept with minor additional interfaces. We also discussed the pros and cons of the proposed protocol.

We currently are studying in relieving contention for lock and log data by exploiting the semantics of these data. Also, we are working on the development of DSM adopted object storage system, SOPRANO[1].

References

[1] J.-H. Ahn, K.-W. Lee, and H.-J. Kim. "Soprano: Implementation of High Performance Object Storage System Using Distributed Shared Memory". In preparation, 1995.

[2] R. Ananthanarayanan, S. Menon, and A. Mohindra. "Experiences in Integrating Distributed Shared Memory with Virtual Memory Management". *ACM Operating System Reviews*, 26(3):4–26, July 1992. vm-dsm-expr.ps.Z.

[3] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. "Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence". In *Proc. 1990 Conf. Principles abd Practice Parallel Programming*, pages 168–176, 1990.

[4] M. J. Carey, M. J. Franklin, and M. Zaharioudakis. "Fine-Grained Sharing in a Page Server OODBMS". In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, Minneapolis, Minnesota, May 1994.

[5] R. G. G. Cattel. *"Object Data Managemnet"*. Addison Wesley, 1991.

[6] W. Effelsberg and T. Haerder. "Principles of Database Buffer Management". *ACM Trans. Database Syst.*, 9(4), Dec. 1984.

[7] J. L. Eppinger, L. B. Mummert, and A. Z. Spector, editors. *Camelot and Avalon: A Distributed Transaction Facility*. Data Management Systems. Morgan Kaufmann Publichers, Inc., San Mateo, CA, 1991.

[8] B. D. Fleisch and G. J. Popek. "Mirage: A Coherent Distributed Shared Memory Design". In *Proc. 14th ACM Symposium Operating System Principles*, pages 211–223, 1989.

[9] A. Forin, J. Barrera, M. Young, and R. Rashid. "Design, Implementation, and Performance Evaluation of a Distributed Shared Memory Server for Mach". Technical Report CMU-CS-88-165, School of Computer Science, Carnegi Mellon University, Aug. 1988.

[10] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman Publishers, Inc., 1993.

[11] T. Haerder and A. Reuter. "Principles of Transaction-Oriented Database Recovery". *ACM Comput. Surv.*, 15(4):287–317, Dec. 1983.

[12] A. B. Hastings. *"Transactional Distributed Shared Memory"*. PhD thesis, School of Computer Science, Carnegie Mellon Univ., 1992. CMU-CS-92-167.ps.

[13] M. Hsu and V.-O. Tam. "Managing Databases in Distributed Virtual Memory". Technical Report TR-07-88, Harvard University, Mar. 1988.

[14] M. Hsu and V.-O. Tam. "Transaction Synchronization in Distributed Shared Virtual Memory Systems". Technical Report TR-05-89, Aiken Computaion Lab. Harbard University, Jan. 1989.

[15] J. E. B. Moss. "Working with Persistent Objects: To Swizzle or Not to Swizzle". *IEEE Trans. Softw. Eng.*, 18(8):657–673, Aug. 1992.

[16] B. Nitzberg and V. Lo. "Distributed Shared Memory: A Survey of Issues and Algorithms". In T. L. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 375–386. IEEE Computer Society Press, 1994.

[17] I. L. Traiger. "Virtual Memory Management for Database Systems". *ACM Operating System Reviews*, 16(4):26–48, Oct. 1982.

[18] Y. Wang and L. A. Rowe. "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture". In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 367–376, Denver, Colorado, May 1991.

[19] S. J. White and D. J. DeWitt. "A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies". In *The Proceedings of the International Conference on Very Large Data Bases*, Aug. 1992.