Minimizing Consistency Traffic in a Versioned Object Transactional DSM *

Arne Grimstrup
Department of Computer Science
Dartmouth College
Hanover, NH 03755-3510, USA
arne@cs.dartmouth.edu

Peter Graham
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
pgraham@cs.umanitoba.ca

Abstract

In this paper we describe an extension of the LOTEC (Lazy Object Transactional Entry Consistency) protocol designed to minimize the network traffic required to maintain consistency in a software-only DSM (Distributed Shared Memory). Unlike traditional DSM systems which are targetted for shared memory parallel computing, the protocols discussed in this paper are designed to support distributed persistent object systems. The object environment affords improved semantic analyses that can be exploited to improve the efficiency of consistency maintenance.

This paper contributes by defining a mechanism for reducing consistency traffic by partitioning each object into a number of "chunks" based on method-attribute affinity so that the amount of data transferred in update messages is minimized. Some initial simulation results for the extended protocol are also presented that illustrate the effectiveness of our chunking technique.

Keywords Distributed Shared Memory, Memory Consistency, Persistent Objects, Transactional Execution

1. Introduction

Many DSM systems have been created to simplify the development of parallel programs for distributed memory environments. These systems have met with only limited success due to the extreme focus on performance that is specific to parallel programming. Such systems can, however, also be applied to other problem domains including transparently distributed persistent object systems. To support such environments, special consistency protocols have to be developed. One such protocol for distributed object systems, LOTEC [20], has been designed to support transactional execution of object methods in a DSM.

* This research was supported in part by Natural Sciences and Engineering Research Council grants OGP-0194227.

Any consistency protocol must strive to minimize both the number and size of messages sent in order to be efficient. The LOTEC protocol supports the sending of only the updated parts of objects to save transmission overhead but does not discuss how such parts are to be derived and managed. In this paper, we describe a technique for partitioning objects into groups of distinct attributes based on the expected behaviour of the methods that will operate on those attributes. By partitioning objects in this way, consistency traffic may be effectively minimized for all likely computations performed by an object.

1.1. Organization

The rest of this paper is organized as follows. In Section 2 we review related work. Section 3 discusses the problem and our assumptions. Our solution strategy, object chunking, is described in Section 4 and we discuss our simulation results in Section 5. We present our conclusions and discuss directions for future research in Section 6.

2. Related Work

2.1. Distributed Shared Memory

Over the last 15 years, the development of software Distributed Shared Memory (DSM) systems has been widely investigated. Following the pioneering work of Li and Hudak [15] a large body of knowledge has been created focusing, primarily, on improving the performance of such systems by reducing the number and size of messages sent. This has been accomplished using weak consistency protocols [7, 13, 4] as well as other optimizations [8, 1, 2, 21]. Work has also been done on decreasing the overhead of the transmission protocols used to send consistency maintenance messages [23, 22, 6] and on the underlying networks themselves (e.g. Gigabit Ethernet, Myrinet [5]).

2.2. LOTEC

Sui and Graham [20] described a DSM-based object programming system that uses a transactional execution model based on Moss' closed nested transactions [16]. The consistency of their DSM is maintained using a modified form of Entry Consistency (EC) [4]. Their consistency protocol, LOTEC, uses the synchronization operations defined by the nested object two-phase locking (O2PL) concurrency protocol to drive its operation and the well-defined semantics of the objects operated on are exploited to reduce the amount of data transferred to maintain consistency. LOTEC builds indirectly on earlier work on distributed object systems (including [17, 18]) and objectbases (including [14, 10]). LOTEC is most closely related to work by Fleisch and Hyde [11] and Itzkovitz and Schuster [12].

3. The Problem and Environment

3.1. The Environment

As with LOTEC [20], we assume an environment based on distributed persistent objects. Users interact with the environment by invoking methods on specific objects to accomplish their desired computations. These methods execute as transactions and as such offer the normal benefits of transactional execution: atomicity, consistency, isolation and durability (the so-called "ACID" properties).

Unlike conventional database transactions which are flat, the transactions in this system are nested. Each time a method invokes another method (normally on another object) a sub-transaction is created. This transaction model is an extension of Moss' closed nested transaction model [16] so the updates of sub-transactions are invisible to other transactions until the sub-transaction commits. Further, when a sub-transaction commits, its updates become visible only to its parent transaction (and the parent's children). Only when all the sub-transactions originating from an original user method invocation (the top-level transaction) have successfully completed can the top-level transaction complete. Once a top-level transaction completes, all the updates made by its direct and indirect sub-transactions become visible to other, unrelated transactions.

To ensure the correctness of method executions in such an environment some form of concurrency control must be enforced. Moss defined a version of strict two-phase locking (2PL) [3] that works for closed nested transactions. The LOTEC consistency protocol operates in response to locks acquired and released according to Object 2PL (O2PL) [19] – a modified form of Moss' nested 2PL protocol.

```
INPUT:  OC - the object as a chunk
OUTPUT: CList - list of chunks
CList=CList+OC;
do {
  Changed=FALSE
  forall c in Clist do
    if (BinPartition(c,CLIST))
      Changed=TRUE
} while (Changed==TRUE)
```

Figure 1. Algorithm: `DoChunking`

3.2. The Problem

The lack of a defined technique for partitioning objects is a recognized deficiency of Sui's work [19]. While Sui did explore the benefits of sending only the updated portions of an object during consistency maintenance, she did not consider how objects should be partitioned to enable this optimization. The fact that the operations on an object are well defined (via its class methods) means that techniques can be developed which exploit semantic knowledge of expected access patterns to perform the partitioning. This is the problem addressed in the rest of the paper.

4. LOTEC Extensions

The optimal partitioning of an object into "chunks" would result in the delivery of all the data required for a method's execution (and no more) in a single message. This optimum can seldom be achieved for reasons including partial overlap of attribute sets between different methods and concerns about lost-updates and false sharing. Partitions should be defined based on the attribute that methods' access and the challenge is to find an algorithm that can balance conflicting access patterns to produce chunks that provide good performance in most cases.

The chunking algorithm we now describe is based on the "Optimal Binary Partitioning" (OBP) algorithm of Chu and Ieong [9]. OBP relies on the availability of semantic information about transactions' behaviours. Since transactions correspond to method executions in our environment such semantic information is readily available (via compile-time analysis of class methods). OBP cannot, however, be used directly to partition objects. OBP creates only a binary partition which is insufficient in most cases. Further, while repeated applications of OBP can be used to further partition the set of attributes this can result in situations where communication overheads dominate transmission costs. Finally, the cost function for OBP is inappropriate to the object environment where partitioning must be done.

Our chunking technique works by repeatedly applying a

```
INPUT:  C - chunk to partition
        CList - existing object chunks
OUTPUT: new chunk created? True/False
cagg = cost for all chunks except C
best = cost for all chunks including C
Changed = False
do {
  agg = cagg
  take the next method m from C.mset and
    add it to NewC.mset - the New Chunk
  move m.aset from C.aset to NewC.aset
  forall other methods n in C.mset do
    if (NewC.mset contains n.aset)
      move n from C.mset to NewC.mset
    elif (n.aset intersects NewC.mset)
      add n to NewC.mset
  agg = agg + cost for C and NewC
  if (agg<best) {
    best = agg; Changed = True
  } else {
    return attributes from NewC.aset
      to C.aset
    return methods from NewC.mset
      to C.mset
  }
} while (more untested methods in C)
if (Changed) add new chunk to CList
return (Changed)
```

Figure 2. Algorithm: `BinPartition`

modifed form of OBP (that includes a new cost function) to produce a set of partitions of each object. It is organized as two algorithms shown in Figures 1 and 2 (where a chunk `C` is represented by a set of attributes in the chunk (`C.aset`) and a set of methods which access the chunk (`C.mset`)).

The cost function used in the algorithm `BinPartition` is simply the number of bytes sent for each attribute a in the set of attributes A plus the message overhead calculated using the following formula:

$Cost(A) = Overhead + \sum_{a \in A} sizeof(a)$

5. The Simulations

Object chunking is done to decrease the amount of data sent during consistency operations. In some applications, however, little or no benefit may be realized because attribute access patterns preclude efficient object partitioning. Network overheads (headers, etc.) may also limit the effectiveness of chunking in some situations. To assess the effectiveness of object chunking, we developed a simple simulation system and conducted a series of experiments.

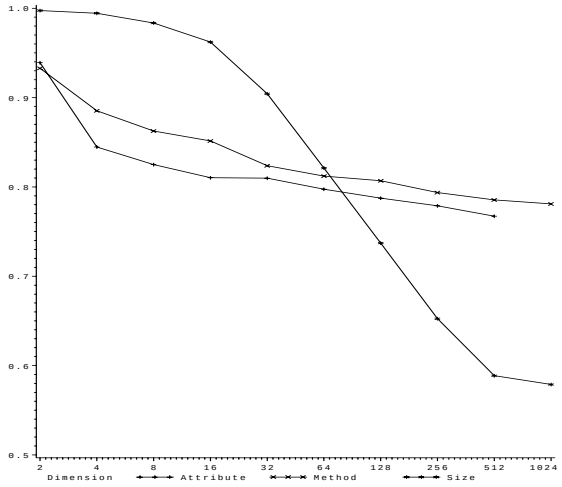

Figure 3. Mean Transmission Cost Ratio vs. Number of Attributes, Methods, and Mean Attribute Size

5.1. The Simulation System

The success of object chunking is tightly tied to the size of objects being chunked and to the attribute usage patterns in class methods. As such, the goal of our simulation was to try to determine when partitioning generates the greatest performance benefit. This was done by examining objects of various size taken randomly from the object space under different usage pattern assumptions.

An object's size is determined by its number of attributes, the size of those attributes, its number of methods, the size of those methods and certain administrative overhead (e.g. object headers). Together with attribute usage pattern, this forms a very large 6-dimensional simulation space which is far too large to enumerate exhaustively.

It is reasonable to assume that object headers, etc. and class methods are replicated at all sites in the system. This eliminates the corresponding transmission overhead and improves performance at a reasonable cost in disk storage. It also effectively removes two simulation dimensions. Because the space remains large, however, our simulations were performed at *representative* points in each dimension.

5.2. The Experiments

We developed a custom, hand-coded simulator to perform the experiments. Sample objects were generated with a fixed number of attributes ranging from 2 to 512 by powers of two. The number of methods per object ranged as the attributes, but was never allowed to be less than the number of attributes in the object nor greater than 8 times the

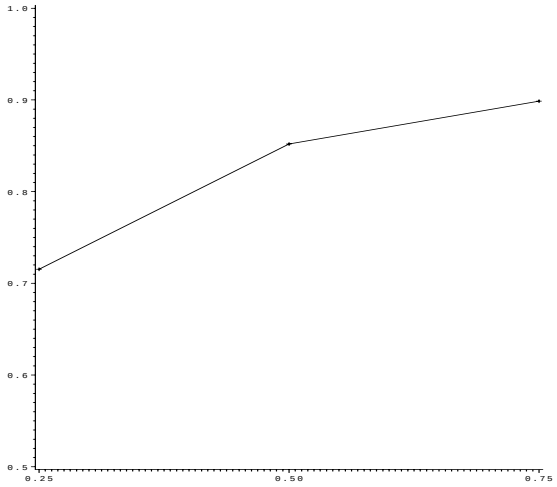

Figure 4. Mean Transmission Cost Ratio vs. Mean Attribute Access Percentage

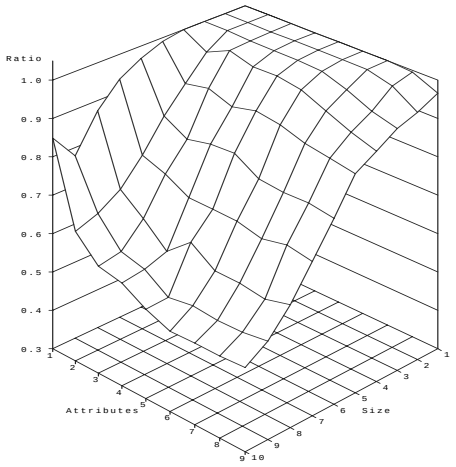

Figure 5. Mean Transmission Cost Ratio vs. Number of Attributes, Methods, and Mean Attribute Size

number of attributes (nor to exceed 1024). The attribute sizes were drawn from a standard normal distribution with a mean value ranging over the powers of 2 from 2 to 1024. The access patterns were generated randomly with the number of attributes accessed drawn randomly from a standard normal distribution with mean value of 25%, 50%, or 75% of the number of attributes in the object.

These values were selected to assess chunking performance in a space that is significantly larger than what we consider realistic (e.g. an object with 512 attributes and 1024 methods would be highly unlikely).

Performance was measured as a ratio of the aggregate transmission cost for the chunked object to the aggregate transmission cost of the unchunked object (including protocol overhead modelled on gigabit Ethernet). The higher the ratio, the less the improvement from chunking the object.

5.3. The Results

We now present selected results from our simulations. These results are summative and generally representative of all results obtained (experimental variance was low).

In Figure 3 the x-axis shows the number of attributes, methods, or the mean size of an attribute in the object and the y-axis gives the mean transmission cost ratio. As expected, the transmission cost ratio improves for chunked objects as the number of attributes, number of methods, and mean size of attributes increase since the number of opportunities to chunk also increases. When attributes are small, regardless of other factors, chunking provides little benefit. This is due to the relatively high protocol overhead assumed and could be addressed using a lower-overhead protocol.

Figure 4 shows the effect of the number of attributes accessed by a method on the transmission ratio. Performance declines as the number of attributes accessed per method increases. This is due to increased overlap between methods which results in multiple chunks having to be loaded.

To obtain further insight into the utility of object chunking, interactions between the various parameters were examined. Of particular interest are the interactions between attribute size and the other parameters. Figure 5 shows the interaction between the number of attributes and the mean attribute size and confirms the conjecture that performance improves as the number of attributes and the mean attribute size decreases. Another interesting feature of chunking is also confirmed: “chunking is unable to improve the transmission costs for objects made up of small attributes”. Intuitively we expect that more chunks will be created in objects with more attributes. The shape of the surface shows that while the number of attributes is important, the resulting chunks must have sufficient size to warrant tha additional transmission overhead to move them.

Similar results were seen when analyzing the interaction between number of methods and mean attribute size and between attribute access percentage and mean attribute size.

The interaction between the number of methods and number of attributes, shown in Figure 6, is also interesting. Here, the results are clearly “stratified” by the number of number of attributes with each larger number showing greater performance improvement. This confirms the intuition that the benefits of chunking increase with object size. The best possible performance is found when the number of methods equals the number of attributes in the object and

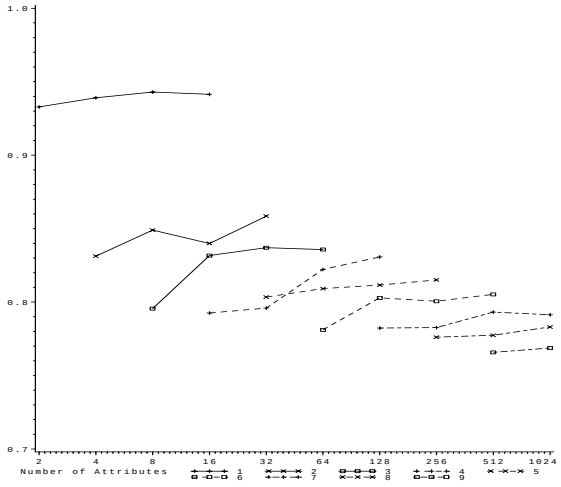

Figure 6. Mean Transmission Cost Ratio vs. Number of Methods Grouped by Number of Attributes

performance degrades as the number of methods increases.

6. Conclusions and Future Work

In this paper we have described a technique for partitioning objects into "chunks". By sending only the useful chunks during consistency maintenace, DSM efficiency is improved. The conditions under which our technique is usefully applicable have also been assessed via simulation.

Immediate future research will include extensions to the simulation system to assess the effect of using network media with different transmission characteristics and an assessment of the concurrency benefits provided by our versioned object model. Longer term research will include integration of the related consistency protocol into our cluster system to assess its effectiveness in a real implementation.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. Software DSM Protocols that Adapt between Single Writer and Multiple Writer. In *Proc. 3rd IEEE Symp. on High-Performance Computer Architecture*, 1997.

[2] C. Amza, A. L. Cox, K. Ramajamni, and W. Zwaenepoel. Tradeoffs between False Sharing and Aggregation in Software Distributed Shared Memory. In *Proc. 6th ACM Symp. on Principles and Practice of Parallel Programming*, 1997.

[3] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The Midway Distributed Shared Memory System. In *Proc. 38th IEEE Int'l Computer Conference*, 1993.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. J. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet - A Gigabit-per-second Local-Area Network. *IEEE Micro*, 15(1), February 1995.

[6] P. Buonadonna. Implementation and Analysis of the Virtual Interface Architecture. In *Proc. ACM/IEEE SuperComputing'98*, 1998.

[7] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proc. 13th Symp. on Operating Systems Principles*, 1991.

[8] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems. *ACM Trans. on Computer Systems*, 13(3), 1995.

[9] W. W. Chu and I. T. Ieong. A Transaction-based Approach to Vertical Partitioning for Relational Database Systems. *IEEE Transactions on Software Engineering*, 19(8), 1993.

[10] O. D. et al. The Story of O^2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 1990.

[11] B. D. Fleisch and R. L. Hyde. High Performance Distributed Objects Using Distributed Shared Memory and Remote Method Invocation. In *Proc. 31st Hawaii Int'l Conf. on System Sciences*, 1998.

[12] A. Itzkovitz and A. Schuster. MultiView and Millipage:Fine-Grain Sharing in Page-Based DSMs. In *Proc. Conference on Operating Systems Design and Implementation*, 1999.

[13] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc. 19th Intl. Symp. on Computer Architecture*, 1992.

[14] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore System. *Communications of the ACM*, 34(10), October 1991.

[15] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, 1986.

[16] J. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, 1985.

[17] T. Ruhl and H. E. Bal. The Nested Object Model. In *Proc. 6th SIGOPS European Workshop*, 1994.

[18] M. Shapiro and P. Ferriera. Larchant-RDOSS: A Distributed Shared Persistent Memory and Its Garbage Collector. In *Proc. 9th Int'l Workshop on Distributed Algorithms*, 1995.

[19] Y. Sui. *DSVM Consistency Protocols for Nested Object Transactions*. MSc thesis, University of Manitoba, 1998.

[20] Y. Sui and P. Graham. LOTEC: A Simple DSM Consistency Protocol for Nested Object Transactions. In *Proc. Intl. Conf. Principles of Distributed Computing*, 1999.

[21] M. Swanson, L. Stoller, and J. Carter. Making Distributed Shared Memory Simple, Yet Efficient. In *Intl. Conf. on High Performance Systems (HIPS'98)*, 1998.

[22] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: AUser Level Network Interface for Parallel and Distributed Computing. In *Proc. 15th Symp. on Operating Systems Principles*, 1995.

[23] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E.Schauser. Active messages: A Mechanism for Integrated Communication and Computation. In *Proc. 19th Intl. Symp. on Computer Architecture*, 1992.