Lazy Home-Based Protocol: Combining Homeless and Home-Based Distributed Shared Memory Protocols

Byung-Hyun Yu, Paul Werstein, Martin Purvis, and Stephen Cranefield

University of Otago, Dunedin 9001, New Zealand
{byu, werstein}@cs.otago.ac.nz,
{mpurvis, scranefield}@infoscience.otago.ac.nz

Abstract. This paper presents our novel protocol design and implementation of an all-software page-based DSM system. The protocol combines the advantages of homeless and home-based protocols. During lock synchronization, it uses a homeless diff-based memory update using the update coherence protocol. The diff-based update during lock synchronization can reduce the time in a critical section since it reduces page faults and costly data fetching inside the critical section. Other than the update in lock synchronization, it uses a home-based page-based memory update using the invalidation coherence protocol. The protocol is called *"lazy home-based"* since the home update is delayed until the next barrier time. The lazy home update has many advantages such as less interruption in home nodes as well as less data traffic and a smaller number of messages. We present an in-depth analysis of the effects of the protocol on DSM applications.

1 Introduction

Parallel programming by means of distributed shared memory (DSM) has many advantages since it can hide data communication between nodes. The ease of programming is in contrast to message passing in which a programmer controls data communication between nodes explicitly. Parallel programming with message passing can be very cumbersome and complicated when a programmer deals with fine-grained data structures. However, the programming convenience in DSM comes with the extra cost of achieving memory consistency over all nodes. In a page-based shared virtual memory system, the extra data traffic for memory consistency becomes worse due to the large granularity of the memory unit, which can cause false sharing. Therefore, it is more challenging to implement an efficient page-based DSM system.

Weak memory consistency models can reduce the data traffic and the number of messages required for memory consistency by relaxing the memory consistency conditions. For example, Entry consistency (EC) [1] and Scope consistency (ScC) [2] models provide the most relaxed memory consistency conditions by taking advantage of the relationship between a synchronization object and data that are protected by the synchronization object. With more relaxed models, more optimized implementations of DSM are possible in terms of less data traffic, fewer messages transferred, less false sharing and fewer page faults, though they create more restrictions on the conditions for a correctly executing program.

2 Homeless Versus Home-Based DSM Protocol

Apart from the relaxed memory models, it is also important to implement the models in an efficient way in order to take advantage of the relaxed constraints that the models can provide. For example, a homeless [3] or a home-based protocol [4] can be used to implement the memory models. In the homeless protocol, the required up-to-date memory is constructed by distributed diffs which contain all the write history of the shared pages. A diff can be identified by its base page number, creating node and vector time-stamp. The vector time-stamps are used in order to apply diffs in the manner of the happens-before partial order. In the home-based protocol, each shared page is assigned a home node. The home node takes responsibility for keeping the most up-to-date copy of the assigned pages. Non-home nodes should send diffs of the page to the home node in order to keep the home pages up to date.

The two protocols have their strengths and weaknesses. In page-based DSM systems, the fine-grained diff-based update used in the homeless protocol has an advantage since unnecessary data in a page are less often transferred. Although it is dependent on an application's memory access patterns, by the nature of the protocol, the homeless protocol has less data traffic since it is not necessary to update a home node at synchronization points. However, a diff in the homeless protocol cannot be removed until all nodes have the diff. This will require more memory space to store unnotified diffs. Ultimately, garbage collection is required when the memory size for the diffs exceeds the predetermined threshold, which is very costly due to global diff exchange. Also diffs created from the same page could be accumulated in a migratory memory access pattern, which makes the homeless protocol less scalable. In terms of coherence-related load balance, the homeless protocol is more susceptible to having one node service requests from many other nodes [5], which is known as a *hot spot.* A hot spot in the homeless protocol makes it more difficult to be scalable.

The home-based protocol can solve much of the scalability problem of the homeless protocol. A diff created can be removed immediately after it is known to the home node, so garbage collection is not needed. There is no diff accumulation since each diff is incorporated into the page of the home node immediately at the synchronization time. Moreover, an efficient implementation is possible with the home-based protocol at synchronization points, with each node updating home nodes with one message containing aggregated diffs created from different pages but belonging to one home node. This optimization reduces the number of messages thus avoiding many send and receive network operations which would be very inefficient. With the same scenario in the homeless protocol, each node just waits until other nodes ask for the diff of a faulting page. In case of multiple writer applications, these diff requests are sent to many nodes which have a diff of the page. This is inefficient compared to one round trip page request in the home-based protocol.

The weaknesses of the home-based protocol are also well known. First, without good home assignment the home-based protocol may suffer. In particular, DSM applications that have regular and exclusive memory access patterns are very sensitive to good home assignment. Second, upon a page fault, even if one byte of data is needed, the whole virtual page, which is 4096 bytes in Linux must be transferred. Therefore, a dynamic home assignment scheme and more fine-grained diff-based update are needed to avoid these weaknesses.

3 Lazy Home-Based Protocol

3.1 Protocol Design

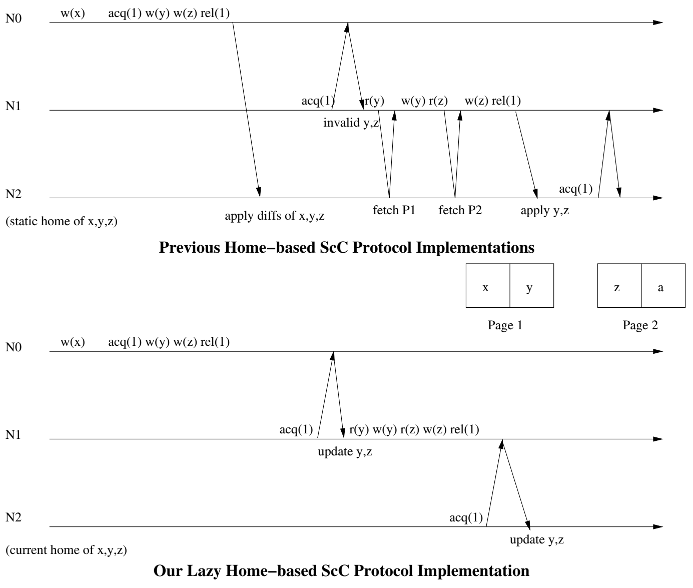
To combine the advantages of the homeless and home-based protocols, we adopt ScC as the memory consistency model and use a hybrid protocol. During lock synchronization, the update protocol is applied but during barrier synchronization, the invalidation protocol is applied. The reason for the hybrid coherence protocol is that during lock synchronization a data access pattern is relatively predictable and fine-grained, but during barrier synchronization, it is more unpredictable and large. Therefore the update protocol can selectively send the data updated inside a critical section. More efficiently, the data are piggybacked with the lock ownership transfer. Also, page faults inside the next critical section of the next lock owner are significantly reduced. During barrier synchronization, stale data are invalidated so that the most up-to-date data can be obtained from a home node.

Compared with previous implementations of the home-based protocol, our protocol is more "lazy" because it does not send diffs at a lock release time in order to update home nodes but delays home update until the next barrier time. To illustrate this laziness, Figure 1 compares previous home-based protocol implementations with our lazy home-based implementation. As seen in Figure 1, our lazy home-based protocol implementation eliminates two home update diff messages sent by N0 and N1 and two page faults in N1 as compared with previous home-based ScC protocol implementations.

Our implementation takes advantage of ScC more aggressively than previous ones in a sense of "laziness". The *lazy* home update is still correct under the ScC model since other nodes should not read diffs made in non-critical sections before the next barrier and sufficient diffs created in critical sections are transferred to the next lock owner by the update protocol.

3.2 Implementation

To implement the LHScC protocol, we distinguished a non-critical section (NCS) diff and a critical section (CS) diff. A NCS diff is a diff created in a non-critical section, and a CS diff is a diff created in a critical section. Intuitively, according to



N0
w(x) acq(1) w(y) w(z) rel(1)
N1
acq(1)
r(y) w(y) r(z) w(z) rel(1)
invalid y,z
N2
acq(1)
(static home of x,y,z)
apply diffs of x,y,z
fetch P1
fetch P2
apply y,z
Previous Home−based ScC Protocol Implementations
x
y
z
a
Page 1
Page 2
N0
w(x) acq(1) w(y) w(z) rel(1)
N1
acq(1)
r(y) w(y) r(z) w(z) rel(1)
update y,z
N2
acq(1)
(current home of x,y,z)
update y,z
Our Lazy Home−based ScC Protocol Implementation

Fig. 1. Difference between Previous Home-based and Our Lazy Home-based Implementations

our memory model, NCS diffs made between two consecutive barriers should be mutually exclusive to one another. NCS diffs are kept until the next barrier when non-home nodes send their non-home NCS diffs to corresponding home nodes. In this way, all NCS diffs are safely preserved at the corresponding home nodes. As for CS diffs, they are sent to the next lock owner during a lock ownership change. Intuitively, the last lock owner before a barrier should have the most up-to-date data that the lock protects. Therefore, the CS diffs from the last lock owners are sufficient to construct the most up-to-date CS data. Upon arrival at a barrier, the last owner of each lock sends its CS diffs to corresponding home nodes unless a node is the home of the CS diffs. In the case that a node owns the same lock consecutively, diffs created from a same page are numbered so that they are applied at the next lock owner in the happens-before partial order.

We developed a *diff integration* technique to reduce data traffic [6]. Since we used fine-grained diff update during lock synchronization, a diff accumulation problem can arise as in the homeless protocol. The diff accumulation problem can be found in a migratory application in which each node writes on the same page in sequence. The diff integration technique incorporates multiple diffs created from the same page during a critical section into one diff, which solves the diff accumulation problem.

To add a dynamic home assignment feature, we also developed a dynamic home assignment scheme [6]. Our scheme is different from others [7,8,9]. Our protocol updates home nodes after all nodes have a knowledge of optimum home locations. This guarantees minimum data traffic related to home page updates by non-home nodes. On the other hand, other dynamic home assignment schemes predict optimum home locations based on previous memory access patterns. This prediction would work well for applications showing regular and coarse-grained memory access patterns without a migratory pattern. However, for applications showing irregular, fine-grained or migratory memory access pattern these schemes do not work since a future memory access pattern would be different from the previous patterns.

4 Performance Evaluation

Our purpose for the performance measurements is to measure the benefits and side effects of LHScC. We chose seven applications to evaluate our LHScC protocol. The applications are obtained from the TreadMarks application suite except PNN which was implemented by us. The problem sizes and sequential execution times of the applications can be found in Table 1. Note that each application tested over the different protocols is identical. We briefly describe the applications.

- **Parallel Neural Network (PNN)** is a parallel implementation of two neural network algorithms: forward and backward propagations. The data set we trained is the shuttle set obtained from the University of California, Irvine machine learning repository. The data set is divided and allocated to each node to be trained in parallel. In the main loop, each node trains part of the data in parallel. Then the local weight changes calculated in parallel previously are summed sequentially through lock synchronization in order to create the global new weight changes. Since each node's local weight changes are transferred and summed, the memory access pattern is migratory during the lock synchronization.
- **Barnes-Hut** is a simulation of gravitational forces using the Barnes-Hut N-Body algorithm. Our Barnes-Hut application uses only barrier synchronization. The memory access pattern of Barnes-Hut is known to be irregular and fine-grained.
- **Integer Sort (IS)** ranks numbers represented as an array of keys by using a bucket sort. Two different implementations of IS were tested— one has

Table 1. Problem Sizes, Iterations and Sequential Execution Times (secs.)

Application	Problem Size	Iterations	Execution Time
PNN	44,000	235	613.56
Barnes-Hut	64k Bodies	3	79.58
IS-B	2^{24}x2^{15}	20	71.61
IS-L	2^{22}x2^{13}	30	16.39
3D-FFT	64x64x64	50	45.10
SOR	4000x4000	50	49.58
Gauss	1024x1024	1023	15.26

only barrier synchronization **(IS-B)** in the main loop and the other has lock and barrier synchronizations **(IS-L)** in the main loop.

- **3D-Fast Fourier Transform (3D-FFT)** solves a partial differential equation using forward and inverse FFTs. In the main loop only one barrier synchronization happens at the end of each loop. The memory access pattern of each loop is regular.
- **Successive Over-Relaxation (SOR)** calculates the average of neighbouring cells' four values (up, down, left and right). The shared matrix is divided into N blocks of rows on which N processors work. Only the values in boundary rows that two nodes share are sent to each other. Therefore the memory access pattern is very regular, coarse-grained and exclusive.
- **Gauss** solves a matrix equation of the form $Ax = b$. In the main loop, only one node finds a pivot element. After finding the pivot element, all the nodes run Gaussian elimination in parallel. The memory access pattern is very regular and exclusive.

To evaluate our protocol efficiency, we compared our lazy home-based ScC DSM implementation (LHScC) with TreadMarks, which is regarded as the state of the art homeless LRC implementation, and our home-based LRC implementation (HLRC). Note that performance comparisons between LRC and EC [10] or LRC and ScC [2] have been presented previously. Adve et al. [10] concluded that there is no clear winner between EC and LRC. Rather, performance difference between them is not because of the model adopted but because of the unique memory access pattern of each application and coherence unit size. However, Iftode et al. [2] concluded that a ScC-adopted DSM implementation showed better performance than an LRC-adopted DSM implementation in applications that have a false sharing memory access pattern inside a critical section. The applications we chose have no false sharing memory access pattern inside a critical section. Therefore, performance improvements under LHScC have nothing to do with reduction of false sharing due to ScC.

Our dedicated cluster network consists of 32 nodes, each one having a 350 MHz Pentium II CPU and running Gentoo Linux with gcc 3.3.2. All nodes are connected with a 100 Mbit switched Ethernet. Each node is equipped with a 100 Mbit network interface card (NIC) and 192 MB of RAM except Node 0 which has a 1 Gbit NIC and 318 MB of RAM. In previous experiments [5], N0 had a 100 Mbit NIC — the same as the rest of the nodes. After we found out that N0 can cause a hot spot due to the nature of the homeless protocol, we replaced the 100 Mbit NIC with a 1 Gbit NIC. The replacement will benefit the homeless protocol most since the home-based protocol is less susceptible to a hot spot.

4.1 Overall Speedups

As can be seen in Table 2, LHScC retains the scalability of the home-based protocol and avoids most of the poorer performances of HLRC for SOR and Gauss, even though Gauss over LHScC with 32 nodes is slightly worse than over HLRC. LHScC also avoids the poorer performances of TreadMarks in PNN,

Table 2. Comparison of Speed-ups between TreadMarks (TM), Home-based LRC (HLRC) and Lazy Home-based ScC (LHScC)

Apps	4 nodes			8 nodes			16 nodes			32 nodes		
	TM	HLRC	LHScC	TM	HLRC	LHScC	TM	HLRC	LHScC	TM	HLRC	LHScC
PNN	3.9	3.9	3.9	7.1	6.9	7.5	8.8	9.5	13.0	4.8	8.2	17.2
B-H	2.1	2.0	2.1	2.4	2.4	2.7	1.8	2.7	3.1	1.5	3.0	3.4
IS-B	3.4	3.6	3.6	4.1	6.0	5.6	2.5	8.3	7.3	0.9	7.7	6.9
IS-L	2.3	2.9	2.7	1.4	2.6	2.9	0.5	1.6	2.0	0.1	0.9	1.1
FFT	1.4	0.9	1.3	2.1	1.2	2.0	2.9	1.9	2.9	2.9	3.2	3.2
SOR	3.4	1.9	3.2	6.0	2.4	5.4	11.3	3.4	9.4	15.5	3.9	13.4
Gauss	2.1	0.2	1.3	1.6	0.3	1.0	0.9	0.5	0.7	0.4	0.5	0.4

Barnes-Hut, IS-B and IS-L over more than 16 nodes. In particular, LHScC has shown significantly better performance with applications showing coarse-grained migratory memory access patterns in a critical section such as PNN and IS-L.

LHScC showed better scalability compared with the homeless protocol in TreadMarks. For example, over 4 nodes LHScC has no clear performance superiority over the other two protocols. However, over 32 nodes, the LHScC implementation was 3.6 times, 1.7 times, 7.7 times, and 11 times faster than TreadMarks in PNN, Barnes-Hut, IS-B and IS-L, respectively, and 3.4 times faster than HLRC in SOR.

The better performance of SOR and Gauss in TreadMarks can be explained by the lazy diffing technique [3]. The lazy diffing technique, which does not create a diff unless it is requested, is very effective for single writer applications without migratory memory access patterns. But this low protocol overhead is only applied to applications such as SOR or Gauss which have a very regular and exclusive memory access pattern. Migratory applications such as PNN and IS showed much better performance under the home-based protocol. In particular, significant improvements can be achieved in PNN and IS-L under LHScC by diff integration and efficient implementation of the update protocol during lock synchronisation.

The super slow-down shown in PNN, Barnes-Hut and the two IS over a large number of nodes in TreadMarks proves that the homeless protocol is vulnerable to the scalability problem. Up to 8 nodes, the homeless protocol showed relatively comparable performance with the two home-based protocols. However over 32 nodes, TreadMarks showed rapid slow-down except for FFT and SOR. A hot spot, garbage collection and diff accumulation are three major hindrances to scalability in the homeless protocol [5]. SOR and Gauss have the most benefit from the dynamic home migration technique, even though Gauss over 32 nodes showed the adverse effect of the technique due to its frequent barrier synchronisation use.

Generally, Table 3 indicates that the larger the data traffic, the poorer the performance, as strongly suggested in the two IS applications over TreadMarks, and SOR over HLRC. The exceptions are the difference between PNN over TreadMarks and HLRC, and Barnes-Hut over TreadMarks and the two home-based protocols. The reason for the exception of PNN is a frequent occurrence of a hot spot. The reason for Barnes-Hut is that even though it produced less data

Table 3. Comparison of Data Communication Traffic between TreadMarks (TM) and Two Home-based Protocols — HLRC and LHScC

Apps.	16 Processors						32 Processors					
	Number of Messages(K)			Amount of Traffic(MB)			Number of Messages(K)			Amount of Traffic(MB)		
	TM	HLRC	LHScC	TM	HLRC	LHScC	TM	HLRC	LHScC	TM	HLRC	LHScC
PNN	114.2	159.4	102.4	277.6	191.2	97.2	228.1	325.8	204.7	1011.0	403.9	190.3
B-H	2809.5	413.4	594.7	622.9	891.9	858.5	8696.5	792.2	1142.3	1211.0	1635.8	1595.4
IS-B	62.0	65.5	66.8	604.3	118.7	79.7	200.5	158.9	159.1	2327.6	285.6	166.0
IS-L	23.9	26.3	26.3	247.4	44.8	44.2	68.7	54.3	54.2	987.6	93.6	91.4
FFT	134.5	152.2	159.5	223.1	612.5	219.4	320.7	408.4	323.8	453.9	838.6	441.3
SOR	45.5	57.0	69.1	65.6	221.8	87.8	63.1	148.2	96.6	76.0	297.2	118.5
Gauss	120.7	321.5	123.7	124.4	1502.0	130.1	250.4	950.3	253.9	259.2	1756.0	263.7

traffic in the homeless protocol than in the home-based protocol, the number of messages produced was much more, for example nearly ten times more for 32 nodes compared with the home-based protocol. This shows that write-write false sharing applications such as Barnes-Hut over the homeless protocol will produce many diff requests sent to many nodes in order to construct the up-to-date copy of an invalid page. On the contrary, with the home-based protocol, only one page request sent to a home node of the page is sufficient to have the up-to-date copy of the page, which is much more efficient.

In the case of IS-B over 16 nodes, even though TreadMarks produces fewer messages compared to the other two systems, it produces much more data traffic—more than 7.6 times compared with LHScC. The data traffic statistics of IS-B over TreadMarks means that the average size of a message is quite large due to diff accumulation. For example, the average size of a packet in TreadMarks over 32 nodes is 11605 bytes, compared to 1043 bytes in LHScC. This data shows that in IS-B over TreadMarks, when a node requests the required diffs of a stale page from the last writers, the diffs received can be large due to diff accumulation.

To better identify the benefits of LHScC, we measured the times taken during lock synchronization (lock acquire and release) and critical sections in PNN over the three different protocols. As shown in Figure 2, Node 0 (N0) over TreadMarks suffers the most from lock contention due to a hot spot in N0. The hot spot in N0 occurs since only N0 writes on the shared pages at the end of barrier in each loop and those pages are accessed after the barrier by all nodes. Therefore all nodes request the pages from the last writer (N0) at the same time, which makes a hot spot. The hot spot becomes worse due to a migratory access pattern in PNN causing diff accumulation. Meanwhile, a hot spot is removed in HLRC since the shared pages are assigned evenly to the nodes. In PNN pages 1 to 4 are most written by all nodes. Since home assignment in HLRC is statically performed, nodes 1 to 4 share the responsibility of sending the most up-to-date four pages. However, eager home update after lock synchronization interrupts the main computation in nodes 1 to 4. Also nodes 1 and 2 which are two lock manager nodes have another interruption due to lock requests from other nodes. That is why nodes 1 to 4, and in particular, nodes 1 and 2, have relatively long lock synchronization times as shown in Figure 3 (note that the vertical scale in

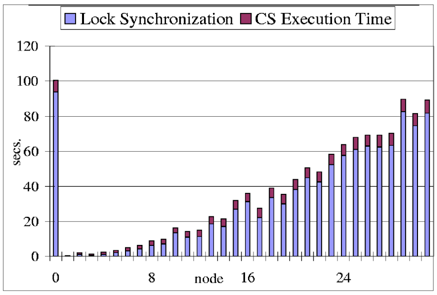


Fig. 2. Lock Synchronization and CS Times in PNN over TreadMarks

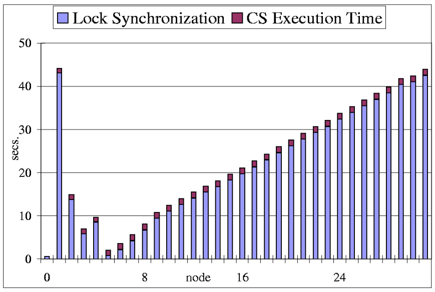


Fig. 3. Lock Synchronization and CS Times in PNN over HLRC

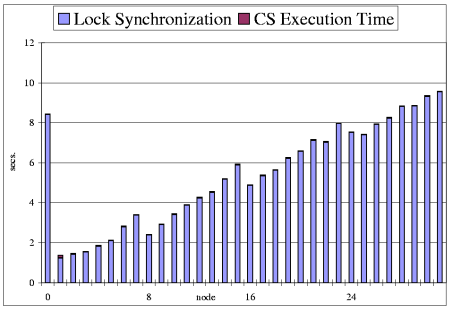


Fig. 4. Lock Synchronization and CS Times in PNN over LHScC

Figure 3 is different from that in Figure 2). Finally, LHScC greatly reduces the time taken for lock synchronization and critical section execution compared with the two other protocols as shown in Figure 4 (again, the vertical scale is reduced). The critical section execution time in LHScC is negligible, just 0.04 seconds on average, compared with 4.46 seconds and 1.29 seconds in TreadMarks and HLRC, respectively. In PNN over LHScC, N0 is a hot spot node as in TreadMarks, but this time the hot spot effect is weakened thanks to our diff integration technique.

5 Related Work

As far as we know, there are two all-software ScC DSM implementations similar to LHScC: Brazos and JiaJia. JiaJia [11] employs a home-based protocol. On the other hand, Brazos [12] is essentially a homeless DSM system. There are also several LRC DSM implementations that have similar ideas to LHScC. Below, we present the comparisons between those systems and LHScC.

Brazos is a homeless page-based all-software ScC DSM system. In Brazos, stale data are made up-to-date by receiving diffs from other nodes efficiently by exploiting multicast communication, compared to LHScC that uses both diffs and pages in order to update stale data without multicast support. Since Brazos uses multicasting for memory coherence during lock and barrier synchronisation, it reduces many complexities of implementing a ScC homeless DSM system. Even though Brazos claims that it uses an adaptation technique to update stale data between homeless and home-based protocols, it is dependent on a page's memory access pattern, and it still has to pay the adaptation and page ownership finding overheads since it uses essentially a homeless protocol. On the contrary, LHScC is much more efficient in combining the two protocols, as it is essentially a home-based protocol with the lazy home update and there is no overhead of combining the two protocols.

JiaJia is a home-based all-software ScC DSM system. However, it has no concept of the lazy home update and only uses the write-invalidate coherence protocol. Also, the implementation of ScC differs between JiaJia and LHScC. In JiaJia, a lock manager manages ScC coherence information so that the manager determines which pages should be invalidated for the next lock owner, whereas in LHScC each local node determines the required diffs for the next lock owner. In this way, LHScC can prevent an added burden on a lock manager and is a more fine-grained implementation of ScC. The implementation of ScC in JiaJia is not only inefficient compared to LHScC but also cannot prevent write-write false sharing inside a critical section due to the use of the write-invalidate protocol and the large page granularity.

ADSM [13] is a homeless all-software LRC DSM system in which two adaptations between single and multiple writer protocols, and write-update and write-invalidate protocols, are selectively used based on a page's memory access pattern. Basically it uses the invalidation protocol. However the update protocol is used for migratory pages inside a critical section and producer/multiple consumer pages during barrier synchronisation. Compared to the update protocol used in LHScC, CS data transferred by the update protocol in ADSM are limited to migratory pages only. Also the granularity of the update is the size of a page, whereas there is a more fine-grained diff size in LHScC.

There have been similar ideas of using the two coherence protocols selectively in order to implement a more efficient coherence protocol in a software DSM system. In KDSM [14], instead of using only the invalidation coherence protocol as in most home-based systems, the update coherence protocol is also used only at lock synchronisation times to solve an inefficient page fetch process occurring in a critical section. However, the efficiency obtained by their implementation is still limited due to the LRC model implementation. For example, at the time of release a node has to send modified data to the corresponding homes, which is unnecessary in LHScC. Also, in the case of diff accumulation, the efficiency of their protocol can be severely diminished.

Another similar use of a hybrid protocol is found in the Affinity Entry Consistency (AEC) system [15] even though the AEC system employs the homeless protocol only. In their system, a Lock Acquirer Prediction (LAP) technique is used to predict the next lock owner in order to prefetch required CS data to the next lock owner. We believe that the LAP technique is not required for most lock-based DSM applications since the next lock owner is already determined many times before the release time. When the next lock owner is not determined at the release time, employing the LAP technique leads to unnecessary updating if the prediction is wrong. Rather than updating eagerly based on prediction, it would be better to wait until the next lock owner is determined. In a similar scenario, at the release time, our implementation first creates diffs modified inside a critical section but waits until the next lock owner is determined. Upon receiving a lock request, the diffs previously created and stored are sent to the lock requester with the lock ownership. That is, LHScC eagerly creates and stores required CS diffs, but can lazily transfer those diffs when the next lock request is received after the diff creation.

Finally, Orion [16] is a home-based LRC DSM system. It has a different approach to other adaptation techniques. It exploits a home node which collects all data access information from other non-home nodes. When other non-home nodes are detected as the frequent readers of its home pages, the home node notifies all non-home nodes about the frequent reader nodes. Next, when a non-home node sends diffs to the home node, it also updates the frequent reader nodes, hoping that the diffs are accessed by them.

6 Conclusions

In this paper, we presented the design and implementation of our novel lazy home-based ScC protocol (LHScC). LHScC is different from a conventional home-based protocol in that home update time is delayed until the next barrier time. LHScC combines diff-based update in the homeless protocol and page-based update in the home-based protocol. Our implementation of LHScC includes a dynamic home assignment scheme and a diff integration technique in order to solve wrong home assignment and diff accumulation problems, respectively. Our performance evaluation shows that LHScC retains good scalability of the home-based protocol and removes a static home assignment problem.

References

1. Bershad, B., Zekauskas, M., Sawdon, W.: The Midway distributed shared memory system. In: Proc. of the IEEE Computer Conference (Compcon). (1993) 528–537
2. Iftode, L., Singh, J.P., Li, K.: Scope consistency: A bridge between release consistency and entry consistency. In: Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'96). (1996) 277–287
3. Keleher, P., Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: Treadmarks: Distributed shared memory on standard workstations and operating systems. In: Proc. of the Winter 1994 USENIX Conference. (1994) 115–131
4. Zhou, Y., Iftode, L., Li, K.: Performance evaluation of two home-based lazy release consistency protocols for shared memory virtual memory systems. In: Proc. of the 2nd Symp. on Operating Systems Design and Implementation (OSDI'96). (1996) 75–88

5. Yu, B.H., Huang, Z., Cranefield, S., Purvis, M.: Homeless and home-based lazy release consistency protocols on distributed shared memory. In: Proceedings of the 27th Conference on Australasian Computer Science, Australian Computer Society, Inc. (2004) 117–123
6. Yu, B.H., Werstein, P., Cranefield, S., Purvis, M.: Performance improvement techniques for software distributed shared memory. In: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS 2005), IEEE Computer Society Press (2005)
7. Hu, W., Shi, W., Tang, Z.: Home migration in home-based software DSMs. In: Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99). (1999)
8. Fang, W., Wang, C.L., Zhu, W., Lau, F.C.: A novel adaptive home migration protocol in home-based DSM. In: Proc. of the 2004 IEEE International Conference on Cluster Computing (Cluster2004). (2004) 215–224
9. Cheung, B., Wang, C., Hwang, K.: A migrating-home protocol for implementing scope consistency model on a cluster of workstations. In: The 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada, USA. (1999) 821–827
10. Adve, S.V., Cox, A.L., Dwarkadas, S., Rajamony, R., Zwaenepoel, W.: A comparison of entry consistency and lazy release consistency implementations. In: Proc. of the 2nd IEEE Symp. on High-Performance Computer Architecture (HPCA-2). (1996) 26–37
11. Hu, W., Shi, W., Tang, Z.: JIAJIA: An SVM system based on a new cache coherence protocol. In: Proc. of the High-Performance Computing and Networking Europe 1999 (HPCN'99). (1999) 463–472
12. Speight, W.E., Bennett, J.K.: Brazos: A third generation DSM system. In: Proc. of the USENIX Windows NT Workshop. (1997) 95–106
13. Monnerat, L., Bianchini, R.: Efficiently adapting to sharing patterns in software DSMs. In: HPCA '98: Proceedings of the The Fourth International Symposium on High-Performance Computer Architecture, Washington, DC, USA, IEEE Computer Society (1998) 289–299
14. Yun, H.C., Lee, S.K., Lee, J., Maeng, S.: An Efficient Lock Protocol for Home-Based Lazy Release Consistency. In: Proceedings of the 3rd International Workshop on Software Distributed Shared Memory System, Brisbane, Australia, IEEE Computer Society (2001) 527–532
15. Seidel, C.B., Bianchini, R., de Amorim, C.L.: The affinity entry consistency protocol. In: ICPP '97: Proceedings of the international Conference on Parallel Processing, IEEE Computer Society (1997) 208–217
16. Ng, M.C., Wong, W.F.: Orion: An adaptive home-based software distributed shared memory system. In: 7th International Conference of Parallel And Distributed System (ICPADS 2000), Iwate, Japan, IEEE Computer Society (2000) 187–194