Adaptive Techniques for Home-based Software DSMs*

Lauro Whately[1], Raquel Pinto[1], Muralidharan Rangarajan[2], Liviu Iftode[2], Ricardo Bianchini[2], and Claudio L. Amorim[1]

[1] COPPE Systems Engineering
Federal University of Rio de Janeiro, Brazil
{whately,raquel,amorim}@cos.ufrj.br

[2] Department of Computer Science
Rutgers University
{muralir,iftode,ricardob}@cs.rutgers.edu

Abstract—

This paper proposes and evaluates Home-based Adaptive Protocol (HAP), a software distributed shared-memory system. HAP performs two key functions that distinguish it from most other distributed shared-memory systems: detection of sharing patterns and behavior adaptation based on these patterns. Detection consists of identifying any change in the sharing pattern of a shared page. Adaptation consists of using a strategy that is specific to the sharing pattern detected to optimize the performance of the system. More specifically, HAP uses updates to maintain the coherence of single-writer pages, which fall under the migratory and producer-consumer sharing patterns. Invalidations are used to maintain the coherence of multiple-writer pages, which can potentially be falsely shared. As part of HAP's adaptation strategy, we dynamically assign homes to pages based on their sharing patterns. We performed preliminary experiments on an 8-node cluster of PCs. Our results show that the current implementation of HAP substantially improves the performance of single-writer applications in which shared pages are modified in critical sections protected by locks. The results also indicate potential improvement in the performance of applications exhibiting other sharing patterns such as producer-consumer, single-writer between barriers. However, the detection and adaptation techniques for these patterns have to be redesigned to exploit the real performance gains that can be achieved with the adaptive system.

Keywords— **Software Distributed Shared Memory, Home-based Lazy Release Consistency, Home-based Adaptive Protocol**

I. Introduction

Research in the last decade has shown that software distributed shared memory (DSM) is a cost-effective method of providing the shared memory abstraction on a network of computers, as it requires virtually no hardware support, can use off-the-shelf operating systems, and delivers good performance for a large class of applications. However, many applications running on software DSM systems suffer high communication and coherence-induced overheads that limit performance [IJ99]. The most common software approach to reducing these overheads is to employ relaxed memory consistency models [CBZ91, KCZ92] to delay and/or restrict communication and coherence operations as much as possible. Another common way of reducing communication and coherence overheads is to employ multiple-writer coherence protocols [CBZ91], which allow two or more processors to modify their local copies of shared data concurrently, merging modifications only when necessary.

Home-based Lazy Release Consistency (HLRC) [ZIL96] is an example of software DSM system that employs both relaxed consistency and multiple-writer coherence. In HLRC, the modifications made locally by each node to a shared page are propagated at certain synchronization points to a home node for the page. Future accesses to the page require retrieving a copy of the page from the home node. HLRC provides good scalability by reducing the number of messages and memory overhead compared to homeless DSM systems, such as TreadMarks [KCZ92].

However, HLRC has some potential disadvantages [CLHW99]: (i) the whole page is transferred on a page fault, even if only one word of it has been modified and (ii) performance can be poor for some applications, if homes are not assigned properly. A poor assignment can significantly increase the number of coherence actions, messages passed, and bytes transferred. Furthermore, HLRC (as well as homeless systems) can be optimized by dynamically adapting its coherence protocol to the sharing patterns exhibited by applications. Such adaptation can reduce the number of required coherence actions in software DSM systems for certain applications [MB98, ACSD+99].

The main goal of our work is to extend HLRC to migrate home nodes and adapt its coherence protocol dynamically, according to sharing patterns. Thus, in this paper we propose and evaluate Home-based Adaptive Protocol (HAP), a software DSM system based on HLRC. HAP performs two key functions that distinguish it from most other software DSM systems: detection of sharing patterns and behavior adaptation based on these patterns. Detection consists of identifying any change in the sharing pattern of a shared page.

*This work was supported in part by Rutgers University and by Brazilian Finep and CAPES agencies.

Adaptation consists of using a strategy that is specific to the sharing pattern detected to optimize the performance of the system. Both detection and adaptation in HAP are inspired by similar features in the homeless ADSM system [MB98].

In more detail, HAP includes the following features:

- Dynamic adaptation between multiple and single-writer coherence protocols;
- Dynamic adaptation between invalidation and update-based coherence – updates are used for single-writer pages, whereas invalidations are used for multiple-writer pages; and
- Home migration of single-writer pages to the writing node.

We currently have a preliminary implementation of HAP. We performed experiments with this implementation on a cluster of eight dual-processor Pentium III-based PCs. We isolated the performance benefits of each of the adaptation strategies and compared our results against the basic implementation of the HLRC system. The results show that the current implementation of HAP substantially improves the performance of single-writer applications in which shared pages are modified in critical sections protected by locks. The results also indicate potential improvement in the performance of applications exhibiting other sharing patterns such as producer-consumer, single-writer between barriers. However, the detection and adaptation techniques for these patterns have to be redesigned to exploit the real performance gains that can be achieved with the adaptive system.

The remainder of this paper is organized as follows. Section II provides an overview of HLRC, serving as background material for the rest of the paper. Section III describes the HAP system. Section IV presents the experimental environment and the application workload. The results are presented in section V. Section VI discusses the related work. Finally, section VI summarizes our work and presents our conclusions.

II. Background

Several software DSM systems use virtual memory protection bits to enforce coherence at the page level. To minimize overheads these systems usually exploit relaxed consistency models and multiple-writer coherence protocols.

One of the most popular consistency models is Lazy Release Consistency (LRC). The LRC algorithm [KCZ92] divides the program execution into intervals delimited by synchronization operations and computes a vector timestamp for each interval. This vector describes a partial order between intervals of different processors. On a lock acquire operation, the last releaser can determine the set of write notices (descriptions of the modifications made to shared data) that the acquiring processor needs to receive, i.e. the set of notices that precede the current acquire operation in the partial order. Upon receiving the notices, the acquirer can then change the state of its memory accordingly.

HLRC is an LRC system that is centered around establishing home nodes for shared pages. The system chooses the first node that accesses a page as the page's home node. In HLRC, a write notice received represents an invalidation that has to be applied to the corresponding page. A future access to an invalidated page requires that the local node send a request for a copy of the page to the page's home node, which always has an up-to-date copy of the page. This scheme works because nodes propagate the modifications they make to shared pages to their corresponding home nodes at lock release points. These modifications are determined using the twinning and diffing mechanism as follows. A page is initially write-protected, so that on the first write to it a violation occurs. On such a violation, an exact copy of the page (a twin) is made and the original copy of the page is made writable. When the actual modifications are required, the twin and the current version of the page are compared to create a runlength encoding of the modifications (a diff).

The lock acquire operation uses a lock manager from which the current owner of the lock is obtained, and a distributed queue where the next acquirers wait for their turn. Barriers are implemented in a distributed way, so that a message with local write notices is sent at a barrier operation to every node. The barrier is completed when the messages with the write notices of all other node are received. More details about the original implementation of HLRC can be found in [ZIL96], where the authors demonstrate the good scalability of the system.

Our current HLRC implementation uses the VI Architecture [Com97] (VIA) standard for user-level communication as its communication substrate. VIA reduces software overhead by avoiding kernel involvement in communication operations. Another important feature of VIA that is used by HLRC is *Remote DMA Write* operations, which allow pages to be fetched from homes with no data copies and diffs to be applied at home nodes without interrupting them. Our implementation of HLRC was described and evaluated in [RI00].

III. Home-based Adaptive Protocol

In this section we describe the sharing patterns detected by HAP, the coherence protocol behavior associated with these patterns, and the details of the transitions between different detected sharing patterns. A longer description of HAP can be found in [Wha01].

A. Access Patterns and Protocol Behavior

The access pattern categorization of HAP is based on the *SPC (Sharing Pattern Categorization)* algorithm created for the homeless ADSM system [MB98]. Each page is categorized as multiple-writer (MW), migratory (MIG), or pro-

ducer/consumer (PC). The migratory pages are subdivided into two groups: MIGi and MIGo. A MIGi page is modified only inside critical sections protected by the same lock. A MIGo page is modified outside of critical sections by only one node in an interval bounded by two consecutive barriers. Note that the pages classified as MIG or PC have a single writer at a time.

A home node can detect the MIGi pattern by checking for lock identification(s) that can come with any diffs it receives. The lock identification specifies the critical section in which the corresponding diffs were created.

The write-notices received at the barrier are used by all nodes to detect the MIGo and PC patterns. Each node detects if there was only one writer to a page, examining the write notices. In that case, the page is categorized as MIGo. If, in the next interval, the single writer is repeated, the page is categorized as PC.

As a MIG or PC page is modified by a single writer, HAP migrates the home of the page to its writer, avoiding the creation of twins and diffs at the writer. The home migration of a MIGi page is done at a lock acquire operation. A list of MIGi pages associated with the lock is maintained, and the lock releaser determines which pages have been modified since the last time the acquirer released the lock. The releaser sends these pages to the acquirer as updates, transferring the homeship of the pages along. After that, the lock and the write notices are sent to the acquirer. Only the releaser and the acquirer have knowledge of the home migration, so a long chain of page request forwards can be generated, which is a potential problem.

The home of a MIGo page is migrated to the node which suffers the first access fault on the page after a barrier. At the next barrier, all nodes will detect the single writer of the page, investigating the write notices. If the single writer matches the first node that accessed the page, HAP has avoided the creation of a twin and a diff. If that is not the case, the home is migrated to the single writer and the creation of the diff at the writer is avoided.

A PC page has only one writer (producer), which is always the same processor, and one or more readers (consumers). The home of a PC page is migrated to the producer, which sends updates to the consumers at the barriers. A consumer does not wait for the updates at the barrier, since frequently there is enough time before the pages are actually accessed by consumers.

The MW pages are handled just like in HLRC, using the twinning and diffing mechanism, invalidation-based coherence, and fixed home nodes.

B. State Transitions

The different page sharing patterns and transitions can be represented by the state diagram shown in figure 1. Initially,

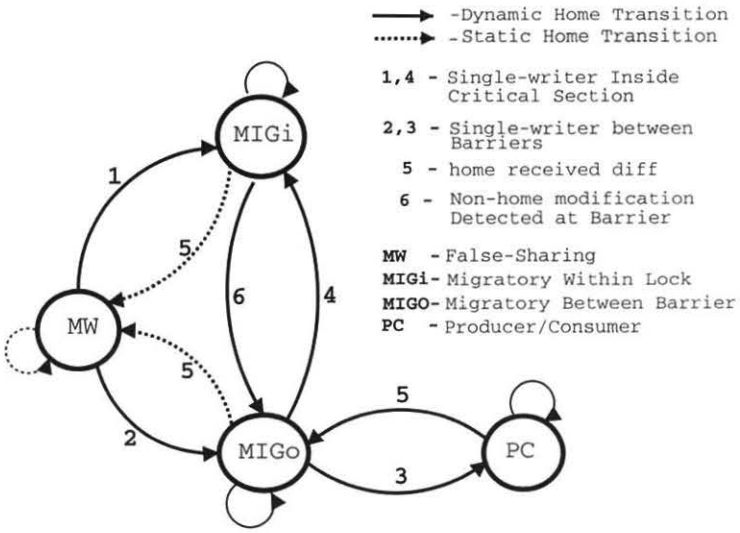

Fig. 1. Page state diagram.

all pages are classified as MW. If the home of a page detects the MIGi pattern, the page enters a transitory state. If the home later receives a page request identified with the same lock, the home will migrate with the page and the page will be categorized as MIGi at the home (transition 1 in the figure). On the other hand, if the home receives a diff associated with another lock, the transitory state is dismissed and the page returns to MW state. At a barrier, the MW state can be changed to MIGo, if the nodes detect a single writer to the page when examining the write notices (transition 2).

A page remains in MIGo state as long as the nodes detect only one writer to the page between barriers. In case the single writer is repeated at two consecutive barriers, the page is categorized as PC (transition 3). The state can change from MIGo to MIGi (transition 4), when a page is modified inside a critical section that is located between two barriers.

A page remains in MIGi, MIGo, or PC state provided that it is only modified by the home. If the home receives any diffs for such a page, its state is "weakened" to MW (transition 5 from MIGi or MIGo) or MIGo (transition 5 from PC).

It is possible for a MIGi page to become a MIGo page (transition 6) when the write notices received at a barrier indicate a single writer to the page that is not the current home.

Finally, note that HAP does not require any messages that are not already part of HLRC either to categorize the pages' sharing patterns or to migrate the pages' homes.

IV. Experimental Environment

We still only have a preliminary version of HAP that contains a first-cut implementation of each of the features described in the previous section. All our experiments with this current version of the system were performed on a cluster of

TABLE I

GIGANET VIA MICROBENCHMARKS

One-way Latency (4Bytes)	8.2 μs
Bandwidth (32KBytes)	101 MB/s
PostSend (4KBytes)	2.1 μs
RegisterMem (4KBytes)	4.3 μs

TABLE II

APPLICATION CHARACTERISTICS

Appl	Problem Size	Synchronization
IS	2^{16} keys, 300 iter.	locks, barriers
SOR	256 x 5120, 100 iter.	barriers
FFT	2^{20} elements	barriers

eight SMP machines. Each machine contains two 650MHz Pentium III processors. However, for this study, we used only one processor at each cluster node. Each processor has a 256KB L2 cache and each node contains 512 MB of main memory. All nodes run Linux release 2.2.14.

Each node has a Giganet cLAN NIC, which is a 32-bit 33 MHZ PCI-based card. These nodes are connected by a 30-port Giganet cLAN switch. The performance characteristics for our experimental platform are reported in the table I. Latency denotes the time to transfer a 1-word packet between two nodes using VIA. PostSend denotes the average time taken to post a send using VIA. The last row presents the cost of the memory registration for communication buffers in VIA.

We use three applications in this study: IS, SOR, and FFT. IS ranks a sequence of integer keys using bucket sort. Its sharing pattern is characterized by migratory pages protected by locks. IS is distributed with TreadMarks. SOR uses the red-black successive over-relaxation method for solving partial differential equations. The black and red arrays are partitioned into groups of rows of roughly equal size, which are distributed among the computational nodes. Communication in SOR involves the boundary rows, which are producer/consumer pages. SOR was developed at the University of Rochester. FFT is essentially the transposition of a matrix of complex numbers that generates an all-to-all, read-based communication. FFT is from the SPLASH2 benchmark suite [WOT+95]. Table II lists the problem size and the synchronization style for each application. We chose to study these three applications because each of them highlights a different aspect of our system.

V. PRELIMINARY RESULTS

In this section we evaluate the performance benefits of each of the main characteristics of HAP, by comparing four versions of our system against HLRC. These versions are: H_MI only supports home migration of MIGi pages; H_MO only supports home migration of MIGo pages; H_PC only supports home migration of PC pages with update coherence; and HAP includes all the mentioned techniques. All these versions are based on the same prototype code for our system.

To better understand where time is currently going, we breakdown the execution time into several categories (averaged over all nodes): computation time, data transfer time, synchronization time, protocol overhead, and time spent in the handler thread, servicing remote requests. Protocol overhead is comprised by diff and twin creation, diff application, write notice handling, and remote request service. Our results are presented in the figures included below. All execution times are normalized with respect to the HLRC execution time.

We also instrumented our systems to collect information about the number of messages and bytes transferred, the number of memory access faults incurred, the number of page requests, and the number of diffs generated. These results are presented in the tables included below. In all cases, we divide the overall statistics by the number of nodes.

A. IS

Figure 2 shows the execution time breakdown for IS. We run IS with HRLC, H_MI, and HAP. We do not present results for H_MO and H_PC because this application is dominated by pages protected by locks. Thus, results for these other versions are the same as for HAP.

As shown in the figure, H_MI and HAP improve performance with respect to HLRC by 19% and 6%, respectively. The main reason for these gains is a significant reduction in lock and barrier overheads (27% in H_MI and 14% in HAP). This reduction is a direct consequence of the optimizations we perform for MIGi pages. Every shared page in IS is classified as MIGi by both H_MI and HAP. The home migration and the update of MIGi pages decrease the coherence overhead and the time spent in the main critical section of the application, which in turn alleviates the serialization effect of the lock and the following barrier.

Table III shows the main execution statistics for each node of IS. As shown in table, H_MI reduces both the number of messages and bytes transferred by 14% in comparison to HLRC. These reductions come from eliminating most diff transfers and a large number of page requests. In terms of access faults, H_MI also behaves better (by 45%) than HLRC, since accesses to MIGi pages do not cause page faults (recall that MIGi pages are transferred during the lock acquire

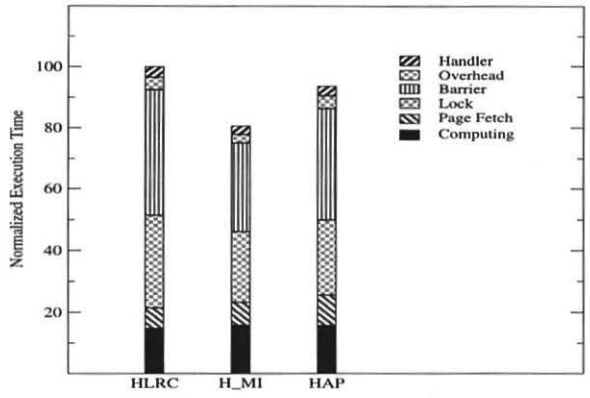

Fig. 2. Execution Time Breakdown for IS

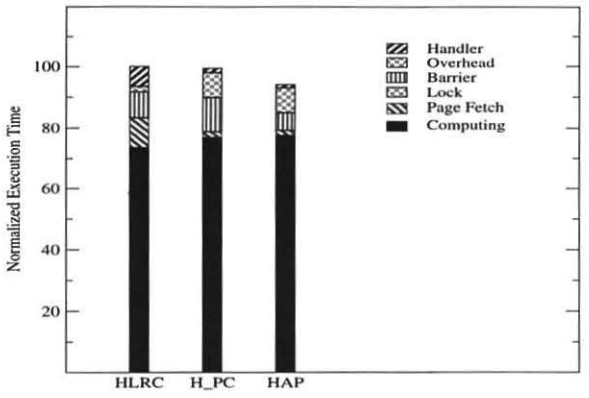

Fig. 3. Execution Time Breakdown for SOR

operation) and writes by home nodes do not cause protection violations.

HAP transfers more messages than H_MI because the MIGi pages are detected as MIGo pages by some nodes. The latter pattern generates more message traffic due to additional write notices and home migration with the first page request. The number of access faults induced by HAP is also greater than that induced by H_MI, since our categorization algorithm requires MIGo pages to be protected more frequently than the other shared pages.

Both H_MI and HAP generate fewer diffs than HLRC because of the special treatment given to single-writer pages, but HAP does so by a much larger margin. The reason HAP generates fewer diffs than H_MI is that under HAP the shared pages are categorized as MIGi by the home node and MIGo by the other nodes. When a non-home node modifies a MIGo page, the diff is not created if the MIGo pattern is confirmed at the barrier point, i.e. the single writer will become the next home node.

B. SOR

The only pages that are effectively shared by more than one node in SOR are PC pages. As shown in figure 3, the optimizations our systems perform for this class of pages are able to reduce the time spent on page requests by 80% (H_PC) and 83% (HAP) and the handler execution overhead by 78% (H_PC) and 83% (HAP), always in comparison to HLRC. Overall, H_PC and HAP outperform HLRC by 1% and 6%, respectively. These reason why these performance improvements are not more significant is the cost of constantly trying to detect MIGo pages in our prototype. This cost reflects itself in higher barrier overheads (H_PC) and higher protocol overheads (H_PC and HAP).

Table IV shows how optimizing for PC pages affects the main execution statistics. In comparison to HLRC, H_PC decreases the number of page requests and messages transferred by 86% and 17%, respectively. The fact that the number of bytes transferred under HLRC and H_PC is roughly the same shows that the updates sent by our system are all useful for this application. HAP exhibits statistics that are similar to those of H_PC.

Note that none of the systems we study involves diff generation for SOR. HLRC uses a "first-touch" mechanism to select the home node for a page. This mechanism is very efficient for this application, since it effectively assigns every producer as the home for the pages it produces, thus avoiding twinning and diffing overheads. HAP and H_PC use the same mechanism to assign the original homes nodes, making migration unnecessary for SOR.

TABLE III
EXECUTION STATISTICS (AVERAGE OVER ALL NODES) FOR IS

	HLRC	H_MI	HAP
Messages (k)	14.4	12.4	14.5
Data (kbytes)	3286.0	2815.2	2697.7
Access Faults	826.4	451.0	676.0
Page Requests	488.8	338.4	338.4
Diffs	262.5	74.7	37.4

TABLE IV
EXECUTION STATISTICS (AVERAGE OVER ALL NODES) FOR SOR

	HLRC	H_PC	HAP
Messages (k)	4.0	3.3	3.3
Data (kbytes)	3770.0	3694.7	3686.2
Access Faults (k)	1.7	1.7	2.0
Page Requests	875.0	120.6	148.3
Diffs	0.0	0.0	0.0

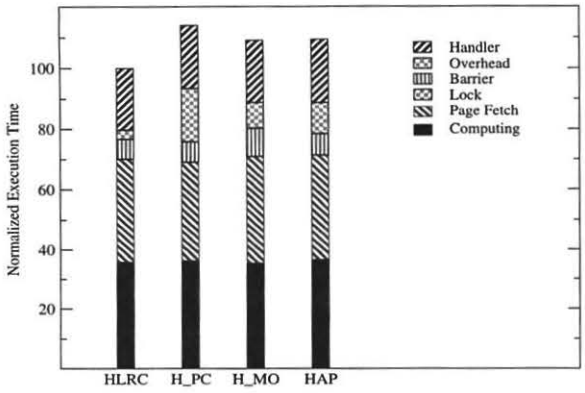

Fig. 4. Execution Time Breakdown for FFT

C. FFT

We present the results for FFT as a worst-case scenario for HAP. As we can see in figure 4, no version of our system outperforms HLRC for FFT. H_PC, H_MO, and HAP degrade the execution time by more than 9%. (H_MI results are not presented because there are no locks in FFT.) The versions that use updates for PC pages (H_PC and HAP) degrade performance because these updates are not really used by the application. As a result, these versions do not reduce the page fetch time and increase the protocol overhead.

Another problem for our systems is that the home assignment scheme of HLRC again works perfectly for FFT. This eliminates any benefits that could be accrued from home node migration. In fact, the MIGo page detection in H_MO and HAP increases protocol overheads and induces an imbalance that affects the barrier times.

Table V shows that the number of access faults and the number of page requests are the same for all systems. In terms of communication traffic, H_PC transfers more messages and data than HLRC, whereas HAP transfers a little less data than HLRC. H_PC transfers more messages and data due to useless PC updates. The reason why HAP performs better than H_PC is that in HAP, before a page becomes PC, it is detected as MIGo. The node that becomes the home of a MIGo page (on its first access to the page after a barrier) is not considered a consumer when the page later becomes PC. As a result, the new home will not receive updates.

TABLE V
EXECUTION STATISTICS (AVERAGE OVER ALL NODES) FOR FFT

	HLRC	H_PC	H_MO	HAP
Messages (k)	5.4	7.2	5.4	5.4
Data (kbytes)	7689.6	11276.5	7574.2	7588.6
Access Faults (k)	3.3	3.3	3.3	3.3
Page Requests (k)	1.8	1.8	1.8	1.8
Diffs	0.0	0.0	0.0	0.0

D. Discussion

We can see from our preliminary experiments that our first-cut implementation of HAP is only successful at improving performance for IS. The optimizations used for the MIGi pages generate a 19% performance improvement in comparison to HLRC. Although our current implementation of HAP did not achieve the same success with the other applications, there is potential for improvements. For example, our techniques decreased the communication overhead and number of page requests in SOR by up to 83% and 86%, respectively. These potential improvements did not translate into substantial gains for the applications we studied for two main reasons: the detection of MIGo pages generates excessive overhead and PC pages are sometimes unnecessarily updated. We believe these problems can be alleviated by replacing the distributed barrier with a centralized one, enabling a better implementation of page categorization and update algorithms. Centralization of the barrier should not degrade synchronization performance for small or medium-scale systems. The categorization algorithm itself should also be modified to make sure that pages are only categorized as PC if both the producer and the set of consumers (not just the producer) are fixed.

VI. RELATED WORK

In this section we discuss the work related to adaptive software DSM systems that we have not yet mentioned. In terms of home-based systems, a few different researchers have proposed home migration. JIAJIA [WST99] dynamically migrates a page's home when a single writer is detected at the barrier. This is the same type of migration performed by HAP for MIGo pages. But that is the only adaptation that JIAJIA does. Chung et al. [CSPP99] and Cheung et al. [CWW99] propose home migration to the processor that suffers the first access fault on a page. All the processors are notified about the new home location by protocol messages. We believe that this approach is excessively aggressive for a variety of applications. HAP is more conservative in that it only migrates a page's home if it has detected that the page has been modified by a single writer. HAP's home migration is used to avoid the creation of twins and diffs without sending any additional messages.

Orion [NW99] does home migration for producer/consumer pages, and also updates the consumers. Both techniques are performed by using broadcasts. In contrast, HAP does home migration for other sharing patterns, and only uses additional messages to update the consumers.

Xie and Han [XH99] proposed a new approach to handle

multiple concurrent writers in software DSM systems called Limited Multiple Writer. It automatically distinguishes two kinds of multiple writers (lock-based and barrier-based) and deals with them with different policies. Lock-based accesses use single-writer coherence and obey LRC, whereas barrier-based accesses can use multiple-writer coherence and obey RC. Adaptation in HAP seems more efficient than this approach, since our system more accurately detects single-writer sharing patterns by separating them into pages modified inside and outside of critical sections.

Keleher in [Kel98] proposes a home-based system that sends updates from a page's producer to its set of consumers at a barrier. This system is shown to outperform HLRC and LRC with updates for regular applications. HAP uses this same technique for PC pages, but also optimizes for the other kinds of sharing patterns.

Two other recent papers have studied adaptation in homeless systems. The Adaptive Striping technique [LYHZ00] extends TreadMarks to automatically alleviate load imbalances. In particular, those that result from multi-paged data structures that have a single writer but multiple readers. Adaptive Striping reduces contention at the writer by evenly distributing the modifications to the other processors at the next barrier. As a result, the writer's effort on behalf of each page that it offloads is limited to constructing and sending the diff to a single processor. This processor is then responsible for servicing the requests for the page from all consumers. This approach is similar to the one used by HAP to update the consumers of PC pages. The principal difference is that HAP updates all the consumers of each page, taking advantage of the low-overhead communication provided by VIA.

Castro and Amorim proposed in [CA01] two techniques called FIESTA and RITMO that detect the pages' sharing patterns with high precision. As a result, they show that better gains are possible by applying adaptive techniques, even to irregular applications. HAP uses a simpler detection technique and can conceivably be improved by using FIESTA and RITMO.

VII. Conclusion

This work introduced HAP, a software DSM system that dynamically adapts to the parallel application's sharing patterns. Adaptation is based on a dynamic categorization of the sharing pattern associated with each page. This categorization is made by a modified SPC algorithm, introduced in the context of the ADSM system, which efficiently categorizes pages as migratory, producer-consumer, and multiple-writer. Based on these sharing patterns, HAP applies the following techniques to optimize performance:

- Dynamic adaptation between multiple and single-writer coherence protocols;
- Dynamic adaptation between invalidation and update-based coherence – updates are used for single-writer pages, whereas invalidations are used for multiple-writer pages; and
- Home migration of single-writer pages to the writing node.

We still only have a preliminary implementation of our system. With this implementation we performed experiments an 8-node cluster of PCs. Our results show that the current implementation of HAP performs well for certain applications, but requires modifications for others.

References

[ACSD+99] C. Amza, A. Cox, L.J. Jin S. Dwarkadas, K. Rajamani, and Willy Zwaenepoel. Adaptive Protocols for Software Distributed Shared Memory. In Proc. of the IEEE, Special Issue on Distributed Shared Memory, Spring 1999.

[CA01] M. Castro and C. Amorim. Efficient Categorization of Memory Sharing Patterns in Software DSM Systems. In *Proc. of the 15th IEEE Int'l Parallel Processing Symposium (IPDPS'2001)*, April 2001.

[CBZ91] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proc. of the 13th ACM Symp. on Operating Systems Principles*, pages 152–164, October 1991.

[CLHW99] A.L. Cox, E. Lara, C. Hu, and W.Zwaenepoel. A Performance Comparison of Homeless and Home-based Lazy Release Consistency. In *Proc. of the 5th IEEE Symp. on High-Performance Computer Architecture (HPCA-5)*, February 1999.

[Com97] Compaq Corporation, Intel Corporation, and Microsoft Corporation. *Virtual Interface Architecture Specification, Version 1.0*, http://www.viarch.org 1997.

[CSPP99] J.W. Chung, B.H. Seong, K.H. Park, and D. Park. Moving Home-based Lazy Release Consistency for Shared Virtual Memory Systems. In *Proc. of the International Conference on Parallel Processing*, Setembro 1999.

[CWW99] B.W. Cheung, C. Wang, and K. Wang. A Migrating-Home Protocol for Implementing Scope Consistency Model on a Cluster of Workstations. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, Junho 1999.

[IJ99] L. Iftode and J.P.Singh. Shared virtual memory: Progress and challenges. *Proc. of the IEEE, Special Issue on distributed Shared Memory*, 87(3):498–507, 1999.

[KCZ92] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc. of the 19th An. Int'l Symp. on Computer Architecture (ISCA'92)*, pages 13–21, May 1992.

[Kel98] P. Keleher. Update protocols and iterative scientific applications. In *Proc. of the 12th Int'l Parallel Processing Symposium (IPPS'98)*, pages 675 – 681, 1998.

[LYHZ00] E. Lara, A. Cox Y.C. Hu, and W. Zwaenepoel. Evaluating the Effect of Contention on Page-Based Software Shared Memory Systems. In *Proc. of Languages, Compilers, and Runtimes for Scalable Computing*, May 2000.

[MB98] L. R. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, pages 289 – 299, February 1998.

[NW99] M.C. Ng and W.F. Wong. Adaptive Schemes for Home-based DMS Systems. In *First Workshop on Software Distributed Shared Memory*, June 1999.

[RI00] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance. In *Proc. of the 3rd Extreme Linux Workshop*, October 2000.

[Wha01] L. Whately. Técnicas de Adaptação para Software DSM Baseado em Residência. *Msc thesis, COPPE/Universidade Federal do Rio de Janeiro*, March 2001.

[WOT+95] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22nd An. Int'l Symp. on Computer Architecture (ISCA'95)*, pages 24–36, May 1995.

[WST99] W.Hu, W. Shi, and Z. Tang. Home Migration in Home-based Software DSMs. In *First Workshop on Software Distributed Shared Memory*, June 1999.

[XH99] X. Xie and C. Han. Adjusting Single-Multiple Writer to False Sharing in Software DSMs. In *Int. Conference on Parallel and Distributing Processing Techniques and Applications*, June 1999.

[ZIL96] Y. Zhou, L. Iftode, and K. Li. 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)*, pages 75–88, October 1996.