Adaptive Schemes for Home-based DSM Systems

M.C. Ng
School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119620
ngmingch@comp.nus.edu.sg

W.F. Wong
School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119620
wongwf@comp.nus.edu.sg

ABSTRACT

Home-based consistency model, a variant of lazy release consistency model (LRC), is a recent development in the DSM field that has open out many areas for further research and development. In this paper, we present 2 adaptive schemes for home-based DSM systems*: home migration* and *dynamic adaptation between write-invalidation and write-update protocols*. The two fully automatic schemes are in line with our objective of building an adaptive DSM system that involves minimal user intervention and yet delivers good performances. We shall present evidence of this by doing a performance evaluation of the two proposed schemes in our software DSM system Orion.

Keywords

Distributed shared memory, virtual shared memory, lazy release consistency model.

1. INTRODUCTION

The advent of multiple writer lazy release consistency (LRC) [3][10] was a major milestone in the development of software distributed shared memory. Since then a good number of variants have appeared. The major challenge facing software DSM is the cost of network communication necessitated by the protocol as well as induced by false sharing. Basically, the cost of network communication can be broken down into two factors:

- the number of messages transacted;
- the amount of data transferred

Of the two, the former is more costly in terms of performance loss. To these we can add a third – the timeliness of data. Software DSMs rely on the use of signal handling mechanisms that are costly. If data is available when it is needed, this can be reduced. This probably explain why in certain applications, write update protocol will perform better than the write invalidate protocol even if more data is transferred.

One of the important variant of LRC is the home-based LRC (HLRC) [1][12]. Unlike the traditional LRC model (which we shall from now on referred to simply as "LRC"), each page in a HLRC has an assigned home. The main advantage of HLRC over LRC is that after communicating diffs to the homes, they can be discarded. In contrast, LRC needs to use an aging mechanism to ensure that diffs do not occupy too much storage. LRC, on the other hand, do not need communicate diffs to the homes thereby potentially reducing communication.

Published data have shown that for certain applications LRC performs better than HLRC [4] while for some other applications, the reverse is true [2]. The main cause for this difference is the differences in the memory access patterns (MAP) exhibited by the applications in question. This points to the need to adapt the protocol to better suit the needs of the applications. Many automatic adaptive schemes [5,6,7,8,9,10,11] have been proposed. These include adaptation between single-writer and multiple-writer protocols, adaptation between write-invalidate and write-update protocols, and even process/thread migration.

HLRC exhibited an important feature that aids adaptation. The homes of pages serve as a natural point from which access pattern information for the pages can be collected. In this paper, we shall focus on taking advantage of this feature. Probably the closest to the adaptive schemes we shall propose is the adaptive migratory scheme proposed by Kim and Vaidya [5]. Their scheme allows each process to independently choose one of the three protocols: migratory, invalidate and competitive update. While their proposals are for general DSM systems, ours are targeted specifically at HLRC by taking advantage of memory access information collection at the home sites. In the rest of the paper, we shall outline our proposals and present some experimental data on their effectiveness.

2. ADAPTIVE PROTOCOLS FOR HLRC DSM SYSTEMS

The choice of the home sites for each page is crucial to performance. Some HLRC systems allow the users to specify how the homes should be distributed before execution. This strategy depends on the user's expertise and the static nature of the application's memory references. This points to the need for good initial home placements and dynamic adaptation.

There are two possible ways to solve the initial home assignment – through user intervention or by means of compiler inference. The latter remains an area for active research. It is the issue of dynamic home placement and migration that we wish to explore. In most modern scientific applications, repetitions or loops of codes are common. This makes the MAP fairly predictable. Hence, knowing when and which process will need the data will contribute to better data distribution and better performance. For shared pages, there can be multiple producers that produces some data shared by either zero, one or many consumers (i.e. demand). We limit our discussion here to one producer. If this supply-and-demand MAP is kept stable, adaptive schemes can easily be formulated. We have categorized the MAP of an application from the *producer's point of view* into 4 main groups:

i) *zero consumer,*

ii) *one persistent consumer,*

iii) *multiple persistent consumer, and*

iv) *non persistent consumer*

Based on these 4 groups of MAPs, we have developed 2 adaptive schemes for HLRC.

	Events	RUC of page *k*	PAC of page *k*	Page *k* access permission	Page request sampling period	Conditions & actions
1	Local access detection (process 0)	Reset all elements to 0	Increment $PAC_{[0]}$ by 1	Change page access permission	Increment by 1	NA
2	Remote diffs update by process *M* (*M*≠0)	Increment $RUC_{[M]}$ by 1	NA	Set page access permission to inaccessible (to detect local access)	NA	If $RUC_{[M]}$ exceeds HMTL, event 4 is triggered off.
3	Page request by process *M* (*M*≠0)	NA	Increment $PAC_{[M]}$ by 1	NA	Increment by 1	If sampling period ends, event 5 is triggered off.
4	Home migration	Reset all elements to 0	NA	Remove read protection	NA	Home is migrated to process *M*
5	Write protocol switch and home migration evaluation	NA	Resets all elements to 0 after evaluation	NA	Reset page request sampling period to 0 after evaluation	If $PAC_{[0]}$ is non-zero and $PAC_{[M]}$ exceeds WPTL, include process *M* is partial update list. If $PAC_{[0]}$=0 and $PAC_{[M]}$ exceeds HMTL, migrate home to process *M*(*only one non-zero PAC value*).

Table 1: System statistics and events

2.1 System Statistics and Parameters

The MAP monitoring procedure, performed at the home process, primarily involves keeping track of some system statistics. The statistics involved in the adaptive actions decision-making process are:

- *Remote update count* (RUC) for each page. It is an array in which each element, *k*, keeps track of the number of times process *k*'s updates have been received since the last local access at home.
- *Page access count* (PAC) for each memory page. PAC includes both local and external access. An external access refers to the remote request for a fresh copy of a page from the home. It is an array with each element pertaining to each remote process.

To detect read operation in the page that is based locally involves additional steps in a traditional DSM system. Typically in a home-based DSM system, pages based locally can either be write-protected or read/write accessible. However, to detect local access for once, the 'home' page has to be made totally inaccessible. The system parameters include:

- *Page request sampling period* for each memory page. Within this sampling period, the PAC of each page is accumulated upon each page request. If the period is set as *Y*, then the Y^{th} page request is considered the end of a period and PAC is reset. Local access is also considered a page request.
- *Home migration triggering limit* (HMTL). This is the limit set to kick off the home migration adaptive action.

- *Write protocol switch triggering limit* (WPTL). This is the limit set to activate the write protocol switch adaptive action.

Table 1 shows a tabular summary of the collection of system information and the conditions that trigger off the adaptive actions. Assume that the events are happening to page *k* whose home is process 0. Note that resetting RUC is essentially to deter unnecessary and extensive home migration.

2.2 Home Migration

It is quite obvious that the HLRC has one serious shortcoming. That is the choice of home for a page. If the initial choice is not good, time and network bandwidth will be wasted. Hence, some DSM systems like home-based Treadmarks do allow the users to specify how the homes should be distributed before execution. It would be even more desirable if the home assignment is automatic and changeable during execution. In designing an efficient HLRC, there are 2 things that we should avoid. They are firstly improper initial home assignment, and secondly static home assignment.

Improper initial home assignment may be sufficiently solved with the help of the user who can specify how homes should be assigned for the start. Yet, the greater hurdle to be overcome is dynamic adaptation to the changing MAP of an application during execution. Static home assignment is never ideal for such an application, and so the solution is home migration.

Home migration is one adaptive scheme that reduces unnecessary network traffic. By monitoring the MAP, we can make intelligent guess of which pages are frequently or seldom fetched by other processes.

2.3 Dynamic Adaptation Between WI and Partial WU protocols

Write protocol switch is an adaptive scheme that improves the timeliness of data availability. It aims at reducing the time spent on waiting for a page to come in from home.

HLRC sends out invalidation messages after diffs are settled at their homes. Note that diffs are simply updates and it is just a convenient extension of the consistency model by sending diffs to all process. Subsequent invalidation messages should therefore exclude pages that are maintained by WU protocol. The question is when to make such a switch. In one of our schemes, if more than 50% of the total number of collaborating processes are in need of that page and each of the non-zero PAC elements exceeds an experimental threshold value (WPTL), then WU protocol is adopted for that page.

We subsequently improved performance further by the introduction of *partial write update* (PWU) protocol. Switching from WI to full WU under the stringent conditions stated above may potentially forgo some opportunities to reduce data transferred. On the other hand, without the stringent conditions, excessive protocol switches can happen. To work around this problem, we use PWU protocol in place of full WU protocol. PWU protocol allows some processes to be updated simultaneously with the home. Therefore, we can relax the condition or parameter that triggers off a protocol switch. A protocol switch occurs when the percentage of PAC by one process, within a sampling period, exceeds a threshold level. This process will be included in an update list that is broadcast to every process. This approach also eliminates sending unnecessary page updates to processes to do not require them in full WU protocol. Note that full WU protocol is actually a subset of partial WU protocol and hence, if the page is not maintained in full WU protocol, it should be included in the invalidation messages as well.

Switching the write protocol of a page from partial WU protocol back to WI protocol is easy. The non-trivial part is deciding when to do it. When a page is in partial WU protocol, processes in the update list do not go to the home to fetch the page anymore, and as such the home does not have enough information to decide if the write protocol of the page should go back to WI. One possible way is the home periodically solicits the local access count (LAC) of a page from other processes. To facilitate local access detection, the pages maintained in partial WU protocol have to be made read-protected as well each time new diffs are applied. Based on the LACs, the home then decides if the write protocol of the page should be changed. If more than 50% of the participated processes have low LACs, then it may warrant for a write protocol change.

HLRC naturally supports partial WU protocol because there is always a home that processes can always rely on for updated versions of pages, and the mandatory send-diffs-and-forget feature during synchronization serve as a foundation for full or partial WU protocol.

2.4 One Producer-Zero Consumer MAP

As an illustration, suppose there are 4 processes and process 0 keeps writing to page *X*. The home of page *X* is process 2. Each time it reaches a synchronization point, diffs are sent to process 2 but page *X* is never or seldom fetched by other processes (including process 2 itself).

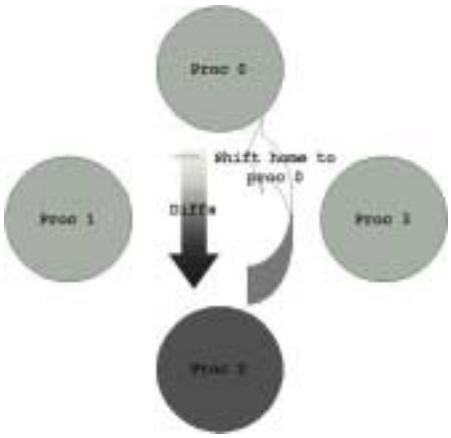

Figure 1: One producer-zero consumer MAP

Clearly, sending diffs of page *X* from process 0 to process 2 is unproductive. Process 2 can decide to shift the home for page X to process 0 if $RUC_{[2]}$ of page *X* exceeds HMTL (i.e. in our system this value is set to 3). Shifting the home to process 1 thus results in lesser redundant network communication.

This MAP is commonly found in applications like SOR in which one process only uses some 'edge' values among the blocks of values computed by other process.

2.5 One Producer-One Persistent Consumer MAP

Consider a slightly different scenario where process 3 persistently requests for fresh copies of page *X* from process 2. Process 2, which does not need diffs from process 0, acts as a buffering zone. Based on the high $PAC_{[3]}$ (i.e. the zero-based PAC value for process 3) and zero $PAC_{[home]}$ (i.e. the PAC value for home) for page *X*, process 2 decides to shift the home of process 3.

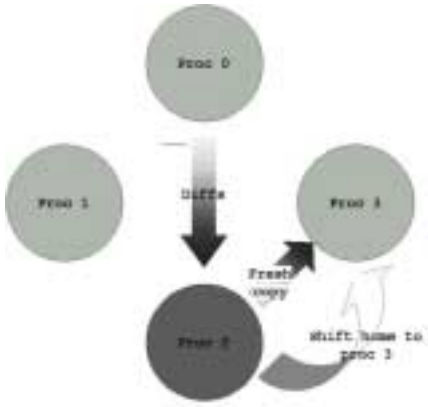

Figure 2: One producer-one persistent consumer MAP

2.6 One Producer-Multiple Persistent Consumer MAP

The one producer-multiple persistent consumer (i.e. more than 1 persistent consumers) MAP should be tackled by dynamic adaptation between WI and full WU protocols since home migration is not suitable in this case.

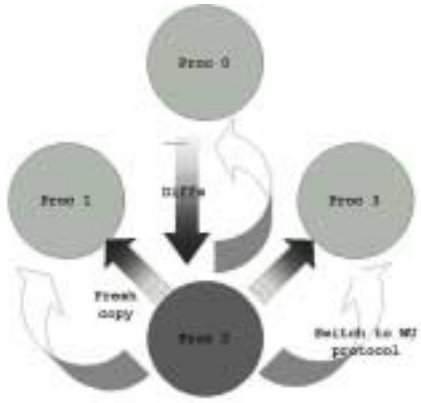

Figure 3: One producer-multiple persistent consumer MAP

Figure 3 shows the scenario where process 0 is the producer while the other processes are consumers. In this case, we make a switch to full WU based on the heuristics described earlier.

2.7 One Producer-Non Persistent Consumer MAP

Lastly, the recommended approach for dealing with the situation of one producer-non persistent consumer is not to do anything, i.e. maintain the page in WI protocol. In fact, this approach is inherently built-in when the system periodically evaluates whether the page should be maintained in WI or partial WU protocol based on the PAC elements of the remote processes. Since most consumers or processes do not consistently request for the page, the average PAC elements will be low. Therefore, no change of write protocol is necessary.

3. IMPLEMENTATIONS

In this section, we describe how to implement the 2 adaptive schemes in a safe manner.

3.1 Home Migration Procedure

When a process decides to shift the home of a page from itself to another process, it sends the entire page with the latest version number to the new home. Any outstanding update messages or page request messages should also be forwarded to the new home during this transient period. Hence, it is possible that the new home may receive diffs or page request sent by itself earlier on. The new home upon receiving the entire page from the old home overwrites its local copy if the page is *not* in read/write accessible state. If it is in the read/write accessible state, it should first extract diffs and adds an entry to the list of modified pages for invalidation purpose before overwriting. The page version number is set to the one given by the old home. The diffs are then applied back to the page as if they were from the remote processes. As usual, any application of diff to the page at home causes the version number to be incremented. The page is then made read-only. As mentioned earlier, the new home may receive the page request message that it sent out. This can be ignored safely. The next thing is to broadcast to every process to declare that it is currently the new home for that page and hence, the old home does not need to forward messages anymore. This completes the home migration procedure.

The least total number of messages involved in this migration procedure is *n-1*, where *n* is the total numbers of processes involved. The first message is from the old home to the new home, and the *n-2* messages to processes other than the old home. The assumption is that there are no outstanding messages directed toward the old home.

3.2 Write Protocol Switch Procedure

When the home of a page decides to switch from WI protocol to partial WU protocol, it should inform all processes to block any access to the particular page. This may be mistaken for accessing an invalidated page. Hence, there should be indication to tell the process to treat this page fault differently. The typical way is to ignore it until the adaptation procedure is complete. Processes receiving the 'freeze' command from the home proceeds to make the page inaccessible and then returns an acknowledgement to home. The message also contains the list of processes in the update list. Note that all processes should still be free to accept and

service any requests so as to prevent deadlock. If this is not observed, simultaneous adaptation processes initiated by different processes may cause a deadlock. When the home receives acknowledgements from *n-1* process, where *n* is the number of collaborating processes, it can conclude that there is no outstanding messages in the network relating to that particular page. The home then sends the entire page to the new members of the update list. The new members then overwrite its local copy if its copy is not in read/write accessible state. If not, it would extract the diffs before overwriting. The diffs are then applied back. The access permission of the page should remain the same before and after the procedure completes. Finally the new members will send back acknowledgement to the home, which then broadcast messages to 'unfreeze' the remote copies. This completes the entire procedure of write protocol switch from WI to WU.

The total number of messages involved during the procedure is given as $3(n-1)+2m$, where m is the number of new members (i.e. $m<n$). Since m is at most $n-1$, the maximum number of messages is $5(n-1)$. Therefore, it is obvious that it is an expensive operation and decision to perform a write protocol switch must be done carefully. Note that there is no need to maintain version number if the process is in the update list.

To switch from partial WU to WI protocol, the home sends 'freeze' messages to all processes. In response, the processes, whether they are members in the update list or not, must extract diffs and send them together with the acknowledgement message back home. At the same time, they may clear the page entry in the list of modified pages to prevent unnecessary invalidation subsequently. Upon receiving all diffs and acknowledgements, the home can conclude that it has the latest version of the page and no outstanding update messages are still in the network. During the transient period, external page request can be ignored. The home may reset its version number and sends 'unfreeze' messages along with a fresh copy of the page to all processes to continue. When the process completes, all processes should have a new version number and possess the latest copy of the page in read-only state. This procedure involves $3(n-1)$ messages.

4. THE EXPERIMENTS

We have tested the proposed schemes using our home-based multithreaded software DSM named *Orion*. Eight benchmarks were used. Barnes-Hut, 3D-FFT and Water are from the SPLASH [13] suite while SOR, GAUSS, IS, CG and MGS are some of the widely used benchmarks in the software DSM community. The data sizes used and some interesting statistics about the benchmarks are shown in Tables 2 and 3.

The hardware platform on which the benchmarking process is carried out is Fujitsu AP3000. It currently supports 32 nodes and each node has a 143-MHz UltraSparc processor. The nodes are connected via a high speed network, APNet. In our benchmarking process, 4 nodes were used.

The software DSM system, Orion, is a home-based DSM system designed to provide POSIX-thread like API to users. The aim is to facilitate the porting of pthread programs to a distributed system. Currently, it provides approximately 50% of pthread's functionalities. Orion was built on MPICH, the public domain message passing interface (MPI) library from Argonne National Lab [20, 21].

The main objective of this benchmarking process is to verify and to show how well the two adaptive schemes can improve the performance. The process does not involve any user intervention in providing clues for the DSM system in initial home assignment. The initial home assignment is round-robin assignment.

Applications	Size (# of iterations)	Synchronization point
SOR	1024X2048 (100)	201 barriers
GAUSS	1024X1024 (1024)	1026 barriers
FFT	64X64X64 (10)	24 barriers
IS	22X16 (50)	250 barriers
WATER	512 (10)	73 barriers 15440 locks
BARNES-HUT	8192	20 barriers
MGS	64X64X64 (5)	124 barriers
CG	14KX14K (10)	792 barriers

Table 2: Sizes and iterations of benchmarks

Applications (executed without adaptive actions)	Execution time (sec)	Amount of data transferred (MB)	No. of msg	No. of page request
SOR	334.008	59	175762	1824
GAUSS	2083.563	670	225399	5513
FFT	133.220	112	18951	4898
IS	201.966	108	24992	8698
WATER	332.490	27	152138	2998
BARNES-HUT	52.760	32	12733	4227
MGS	80.250	50	13729	1477
CG	259.877	90	30779	10615

Table 3: Statistics of benchmarks executed without adaptive actions

4.1 Results

Table 4 summarizes the main performance results of the proposed scheme. As can be seen, compared to the basic home-based DSM system without adaptive action, a speedup ranging from 12% to 94% was observed. Table 5 reports the adaptive actions taken for each benchmark. Figures 4 to 11 shows the number of external requests for every page in each of the eight benchmarks. These requests necessitate network activity as they call for remote page fetches. This is an important characterization of the memory access pattern of each of the benchmarks.

Applications (executed with adaptive actions)	Speedup (%)	Reduction in data transacted (%)	Reduct. in no. of msg (%)	Reduct. in external page request (%)
SOR	58	93	86	38
GAUSS	94	99	88	20
FFT	30	38	-19	-2
IS	21	-31	3	74
WATER	12	26	-7	67
BARNES-HUT	27	-3	-65	15
MGS	59	66	51	11
CG	46	3	17	91

Table 4: Performance results of benchmarks executed with adaptive actions

Applications (executed with adaptive actions)	No. of home migration	No. of write protocol switch
SOR	774	9
GAUSS	390	3
FFT	1079	481
IS	31	60
WATER	221	83
BARNES-HUT	135	284
MGS	493	22
CG	13	37

Table 5: Adaptive actions taken during benchmark execution

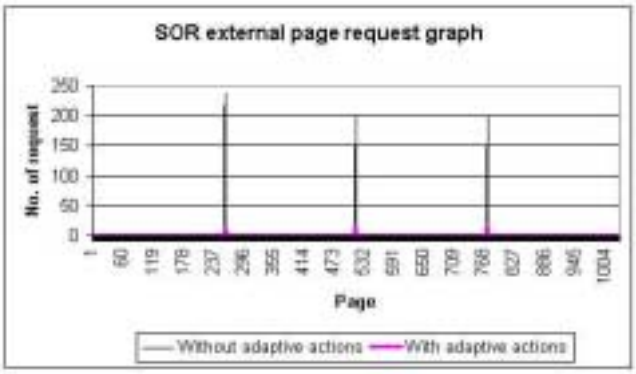

Figure 4: SOR external page request graph

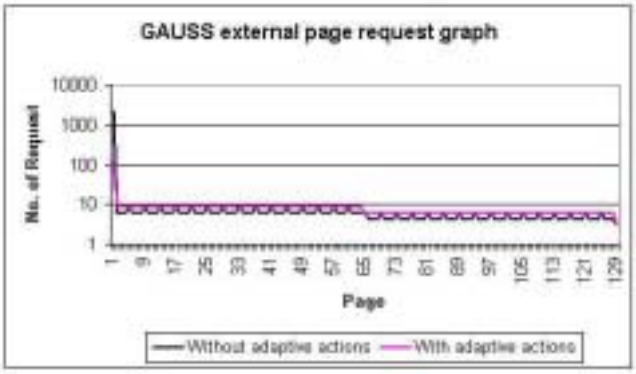

Figure 5: GAUSS external page request graph

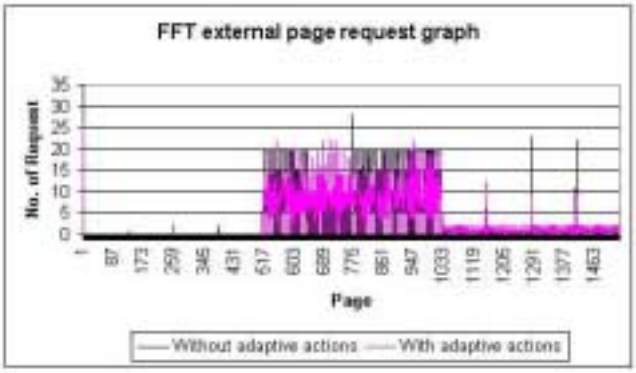

Figure 6: FFT external page request graph

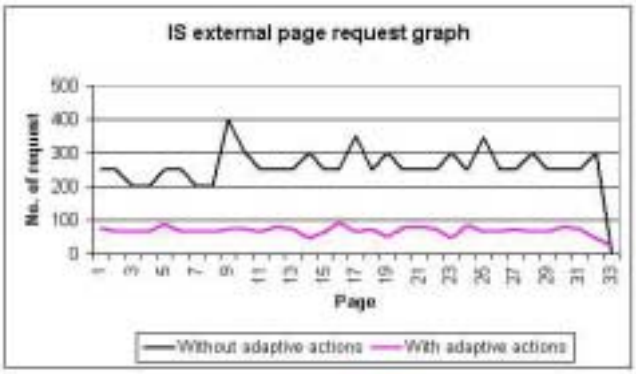

Figure 7: IS external page request graph

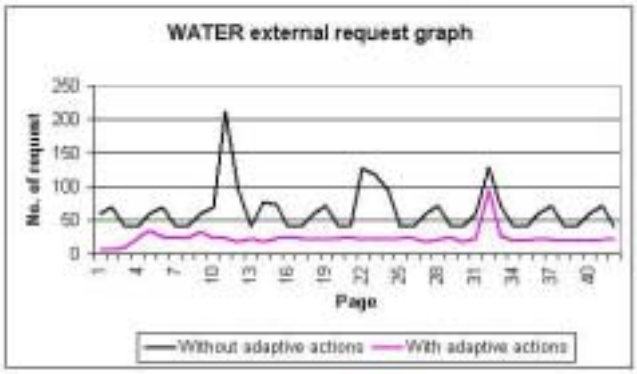

Figure 8: WATER external page request graph

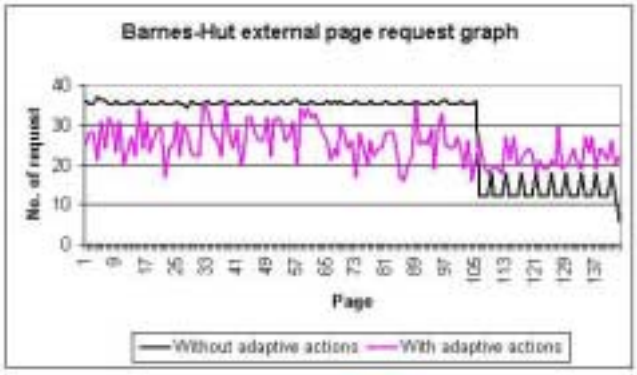

Figure 9: Barnes-Hut external page request graph

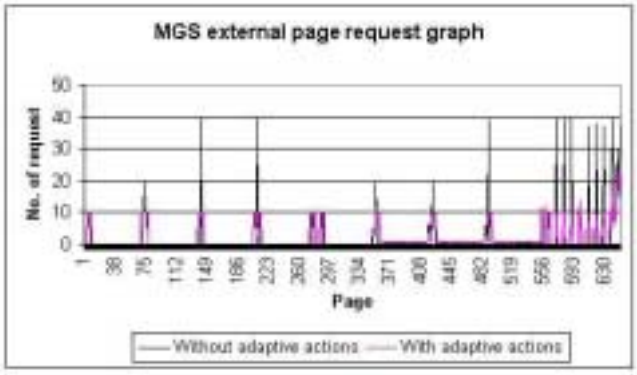

Figure 10: MGS external page request graph

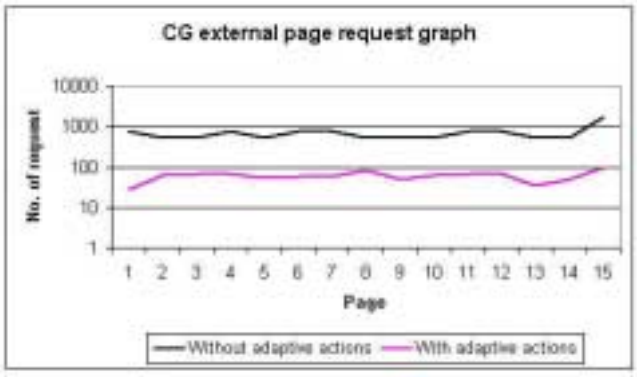

Figure 11: CG external page request graph

In the case of SOR, GAUSS and MGS, reductions in the overall network traffic brought on the expected speedups. However, FFT and IS are also achieving better performance despite increases in amount of data transferred, number of transacted messages, or number of external page requests. In general, a reduction in network traffic should give rise to better performance, but the converse may not be true. Time spent on waiting for diffs or pages can substantially be reduced if they are already available when they are needed. This is the key rationale for page and diff prefetching techniques. The dynamic write protocol adaptation scheme we propose can potentially increase network traffic, as is the case for write update protocols. The primary aim of this adaptation is to reduce external page request. However, this was not the case in FFT, which actually incurs 2% more of external page request. The amount of data transferred reduces by 38% while there are more messages. We attribute this to excessive home migration, a total of 1079 migrations over 1470 pages, leading to more page requests. Admittedly, our understanding of the various key parameters that triggers adaptation is still weak and is a focus of our future work. In any case, FFT still achieved a 30% speedup probably due to the fact that the savings from reducing the amount of data transferred exceeded the cost of the additional messages.

5. CONCLUSION

In this paper, we presented two dynamic adaptive schemes for home-based software distributed shared memory system. They take advantage of the fact that the home of a page is a natural point to maintain memory access pattern information regarding individual pages. Using this information, pages can migrate to new homes dynamically. The second scheme supports partial write-update protocol. For the same page, on those nodes where the page is accessed frequently, the write-update protocol is used, whereas on the others the initial write-invalidate protocol is maintained. Our experiments show that the proposed schemes can effectively reduce the number of external page requests thereby reducing network communication overheads. The proposed schemes are general and can be implemented on any home-based software DSM.

6. ACKNOWLEDGMENTS

We would like to thank Professor Willy Zwaenepoel of Rice University for his kind advice throughout the design and implementation of Orion.

7. REFERENCES

[1] R. Samantha, A. Bilas, L. Iftode, and J. P. Singh. “Home-based SVM Protocols for SMP Clusters: Design and Performance.” Fourth Internal Symposium on High Performance Computer Architecture, February 1998.

[2] Y. Zhou, L. Iftode, and K. Li. “Performance Evaluation of Two Home-based Lazy Release Consistency Protocols for Shared Virtual Memory Systems.” In Proceedings of the Second USENIX Symposium on

Operating System Design and Implementation, pages 75-88, November 1996.

[3] P. Keleher. “Lazy Release Consistency for Distributed Shared Memory.” Ph.D thesis, Rice University, 1994.

[4] A.L. Cox, E. Lara, C. Hu, and W. Zwaenepoel. “A Performance Comparison of Homeless and Home-based Lazy Release Consistency Protocols in Software Shared Memory.” In Proceedings of the Fifth High Performance Computer Architecture Conference, January 1999.

[5] S. Dwarkadas, H.H. Lu, A.L. Cox, R. Rajamony, and W. Zwaenepoel. “Combining Compile-time and Run-time Support for Efficient Software Distributed Shared Memory.” In Proceedings of IEEE, Special Issue on Distributed Shared Memory, pages 476-486, March 1999.

[6] J.H. Kim, and N.H. Vaidya. “Adaptive Migratory Scheme for Distributed Shared Memory.” Technical report 96-023, November 1996.

[7] C. Amza, A.L. Cox, S. Dwarkadas, L.J. Jin, K. Rajamani, and W. Zwaenepoel. “Adaptive Protocols for Software Distributed Shared Memory.” In Proceedings of IEEE, Special Issue on Distributed Shared Memory, pages 467-475, March 1999.

[8] L.R. Monnerat, and R. Bianchini. “Efficiently Adapting to Sharing Patterns in Software DSMs.” In Proceedings of the 4th International Symposium on High Performance Computer Architecture, February 1998.

[9] J.H. Kim, and N.H. Vaidya. “Towards an Adaptive Distributed Shared Memory.” Technical report 95-037, September 1995.

[10] K. Thitikamol and P. Keleher. “Thread Migration and Communication Minimization in DSM Systems.” In The Proceedings of the IEEE, March 1999.

[11] S.J. Eggers, and R.H. Katz. “A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation.” In Proceedings of the 15th Annual International Symposium on Computer Architecture, pages 373-383, May 1998.

[12] C. Amza, A. Cox, S. Dwarkadas, H.H. Lu, R. Rajamony, W.M. Yu, and W. Zwaenepoel. “TreadMarks: Shared Memory Computing on Networks of Workstations.” IEEE Computer, vol. 29 no. 2, pages 18-28, February 1996.

[13] C. Amza, A.L. Cox, S. Dwarkadas, and W. Zwaenepoel. “Software DSM Protocols that Adapt between Single Writer and Multiple Writers.” In Proceedings of the Third High Performance Computer Architecture Conference, page 261-271, February 1997.

[14] D.H. Bailey, J. Barton, T.A. Lansinski, and H. Simon. “The NAS Parallel Benchmark.” RNR technical report RNR-91-002, Januray 1991.

[15] A. Bilas, and J.P. Singh. “The Effects of Communication Parameters on end Performance of Shared Virtual Memory Clusters.” In Proceedings of SuperComputing’97.

[16] A.L. Cox, and R.J. Fowler. “Adapting Cache Coherency for Detecting Migratory Shared Data.” In Proceedings of the 20th Annual International Symposium on Computer Architecture, pages 98-108, May 1993.

[17] W.W Hu, W.S. Shi, and Z.M. Tang. “JIAJIA: An SVM System Based on A New Cache Coherence Protocol.” In Proceedings of the High Performance Computing and Networking (HPCN'99), April 1999.

[18] M.R. Eskiciogiu, T.A. Marsiand, W.W. Hu, and W.S Shi. “Evaluation of JIAJIA Software DSM System on High Performance Architectures.” In Proceedings of the 12th Annual International Symposium on High Performance Computer Systems and Applications, page 76,May 1998.

[19] J.P. Singh, W-D Weber, and A. Gupta. “SPLASH: Standford Parallel Applications for Shared Memory.” Standard University CSL-TR-92-626, June 1992.

[20] W. Gropp and E. Lusk and N. Doss and A. Skjellum. “A high-performance, portable implementation of the {MPI} message passing interface standard.” Parallel Computing, Volume 22, No. 6, pages 789—828, Sep 1996.

[21] William D. Gropp and Ewing Lusk. “User's Guide for MPICH, a Portable Implementation of MPI.” ANL-96/6 Mathematics and Computer Science Division, Argonne National Laboratory, 1996.