Performance Improvement Techniques for Software Distributed Shared Memory

Byung-Hyun Yu and Paul Werstein
Computer Science Department
University of Otago
New Zealand
{byu, werstein}@cs.otago.ac.nz

Martin Purvis and Stephen Cranefield
Information Science Department
University of Otago
New Zealand
{mpurvis, scranefield}@infoscience.otago.ac.nz

Abstract

This paper presents our novel lazy home-based protocol briefly and two implementation techniques. These techniques are developed by exploiting a relaxed memory consistency model such as scope consistency. First, we present a novel diff integration technique which can solve most diff accumulation problems that are prevalent in migratory DSM applications. Second, we propose a dynamic home migration protocol that solves the static home assignment problem in the original home-based protocol. To evaluate our protocol and techniques, we have done performance tests using well known DSM benchmark applications. The performance results proved our diff integration and dynamic home allocation solve the diff accumulation and wrong static home assignment problems respectively.

1. Introduction

Distributed shared memory (DSM) for parallel programming has many advantages over message passing or multiprocessor counterparts. It is more convenient to implement parallel algorithms by using shared variables compared to message passing in which a programmer explicitly sends or receives data between nodes. Compared with multiprocessors, DSM provides cost-effective performance and better scalability.

In spite of these advantages, DSM hasn't been a major attraction to the parallel computing community due to its slow performance. Since DSM should provide a coherent view of memory throughout distributed nodes, it is more challenging to implement an efficient DSM system compared to a message passing system. For example, the large granularity memory unit of a page-based DSM system causes false sharing which produces unnecessary data traffic. Also DSM has more I/O latency since most DSM implementations use network socket functions compared to multiprocessors which use a dedicated I/O bus. This network latency means wasted time for I/O and leads to performance degradation.

To overcome these page-based DSM problems, several memory consistency models have been proposed such as Release Consistency (RC) [1], Lazy Release Consistency (LRC) [9], and Scope Consistency (ScC) [7]. These models enable less data communication traffic and permit a more efficient page-based DSM implementation. In addition, various protocols and techniques have been proposed to alleviate the false sharing problem and reduce network traffic, for example the multiple writer protocol and the diffing technique [10]. Recently a home-based protocol [17] has been proposed that provides a more simple DSM implementation. The home-based protocol showed better performance results and scalability compared to previous homeless protocols [17, 16].

2. Implementation of the ScC model

The ScC model was proposed in 1996, and the model was implemented using the home-based protocol with automatic update network hardware support [7]. Since then, several all-software ScC protocol implementations have been developed, for example, JiaJia [6], JUMP [2], KDSM [11] and Brazos [13]. As far as we know, most ScC protocol implementations use the home-based protocol except the Brazos DSM system which uses the homeless protocol. Previous all-software home-based ScC protocol implementations follow the conventional home-based (CHB) protocol, as shown in Fig. 1, which was first proposed by Zhou et al. [17]. The CHB protocol would have no adverse effect on DSM performance with automatic update network support since the data communication load would be separated from the main computation load. For an all software ScC implementation, however, one must be more careful about data traffic and the number of messages since they will interrupt the main DSM computation, thus leading to performance degradation.

Our novel lazy home-based (LHB) protocol was devel-

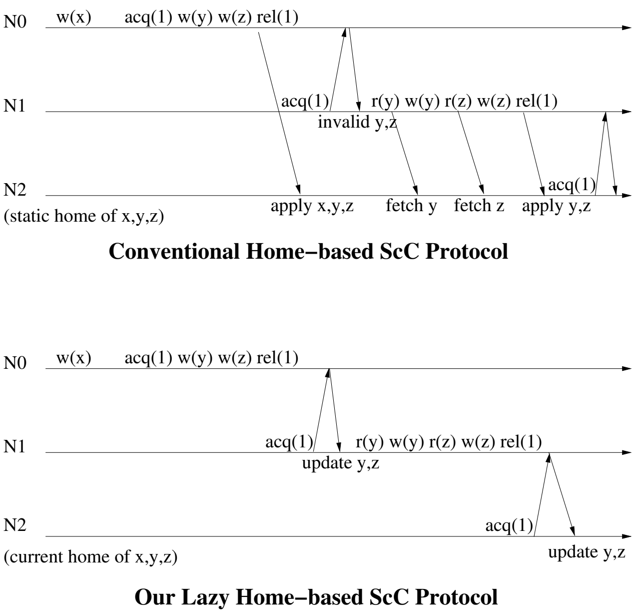

Figure 1. Conventional home-based versus our lazy home-based protocol

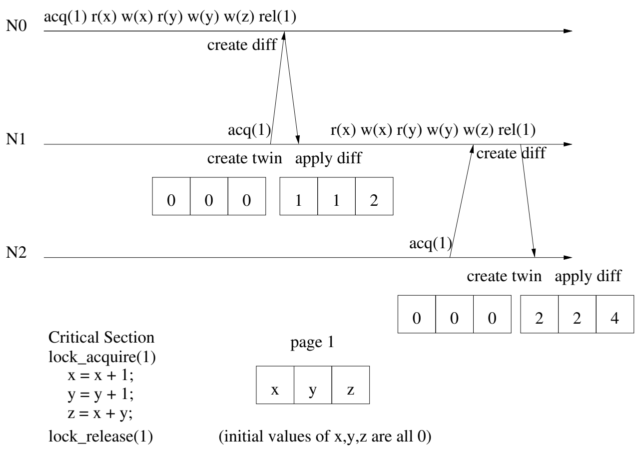

Figure 2. Diff Integration

oped by exploiting the ScC model. Basically the LHB protocol does not send diffs to home nodes between two consecutive barriers. It uses the update protocol during lock synchronization and the invalidation protocol for global scope during barrier synchronization. As seen in Fig. 1, the LHB protocol sends only four messages compared to eight messages in the CHB protocol for the same scenario. In terms of data traffic, it removes home update diffs of node N0 such as $\{x, y, z\}$. It also removes two page faults during the critical section in N1. The only added overhead compared with CHB is twinning and diffing in home nodes. Non-critical section writes (for example, write x in Fig. 1) will be visible to all the nodes after the next barrier. If DSM applications meet the ScC conditions [7], correct executions will be guaranteed.

3. Diff Integration Technique

It is well known that there is a diff accumulation problem in some DSM applications which have a migratory memory access pattern [12]. This problem is found in the homeless protocol but not in the conventional home-based protocol since the conventional home-based protocol updates home nodes as soon as it create diffs. This is one of the reasons why the conventional home-based protocol has shown better scalability for those migratory applications over the homeless protocol [16].

Furthermore, the implementation of the home-based protocol can be better optimized by adopting the ScC memory model as described in Section 2. Previous implementations of the home-based ScC protocol used the invalidation coherence protocol during lock ownership change. However for the ScC implementation, use of the update coherence protocol would be better since the scope is relatively predictable in a critical section which is protected by the same lock. If the update protocol is used to implement the ScC model, the diff accumulation problem occurs even in the home-based protocol. For example, in the lazy home-based protocol in Fig. 1, when N2 requests lock ownership from N1, N1 would send two diffs for page 1 to N2 (a diff forwarded from N0 and the new one created by N1). This diff accumulation seems trivial in this example. However if writes in a critical section are large and executed in many nodes sequentially, the size of accumulated diffs soon become large enough to severely degrade DSM performance.

Our implementation for the diff integration technique is different from previous DSM implementations in terms of the timings of twinning and diff application. Twinning means to create a copy of a page before write access so that tracking the history of writes is made possible. In previous DSM implementations, twinning occurred after diff application and a write page fault since they are only concerned with the host node's writes. For the writes of other nodes, they are updated by distributed diffs in the homeless protocol or pages in the home-based protocol. However twinning in our new technique occurs before diff application and not after a write page fault. In this way, all previous diffs on the same page made in the same critical section are preserved and integrated into a single integrated diff. To understand how our diff integration technique can prevent diff accumulation, Fig. 2 is presented to explain the advantage of diff integration.

As shown in Fig. 2, without diff integration, N1 will send two diffs such as {1,1,2} and {2,2,4} which were made in N0 and N1 respectively to the next lock owner, N2. Also N2 will send three diffs such as {1,1,2}, {2,2,4} and {3,3,6} to the next lock owner. This diff accumulation continues until the next barrier. With the diff integration, N1 and N2 will send only one diff: {2,2,4} and {3,3,6} respectively. The values of the diff represent the up-to-date values of the shared variables written inside a critical section protected by lock 1.

In case of a lock owner acquiring the same lock again, the lock owner accumulates diffs until it receives a lock request and sends accumulated diffs to the next lock owner. A diff created earlier is applied ahead of ones created later at the next lock owner. This local diff accumulation is not a serious problem since diffs accumulated in one node are integrated into single one as soon as they are applied at the next lock owner.

However the diff integration would not be always successful. The previous write history would be lost if a value in a diff created by a node returns back to the same node without any change during lock ownership change and that value is not changed during the current critical section. This scenario is illustrated in Fig. 3.

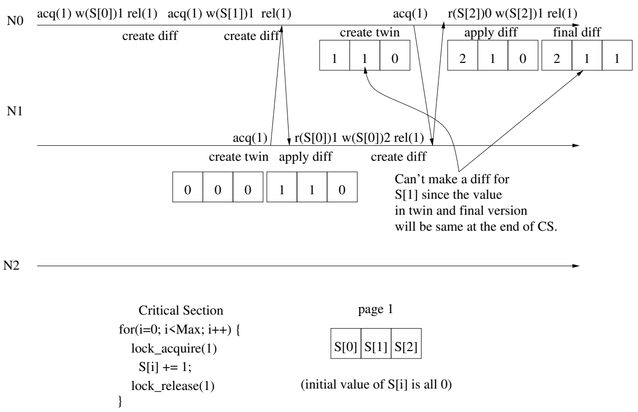

Figure 3. Incorrect Diff Integration

From Fig. 3, it can be seen that a write on S[1] by N0 will be lost after the third lock release in N0 since there is no way to preserve that write. Our current solution to this bad diff integration is that when this situation is detected, that is, when a node owns the previously owned lock again between two consecutive barriers, diff integration stops and diff accumulation happens. Note that from Fig. 3, if N2 acquires the lock from N0 after the third lock release in NO, N2 can integrate accumulated diffs into one diff safely since N2 has not owned the lock before. Many real DSM applications that have the diff accumulation problem do not have reacquisition of previous owned locks. Most migratory DSM applications, in particular iterative scientific applications, own the same lock only once between two consecutive barriers. However, even if reacquisition of previous owned locks is detected so that the diff integration is not applied, it only affects performance, not the correctness of a program execution.

Another solution can be sought from changing the application program itself. For example, if the code in the critical section in Figure 3 is changed into the following code, then there would be no bad diff integration even though the new code would incur more lock synchronization overhead.

```
for(i=0; i<Max; i++) {
  lock_acquire(i);
    S[i] += 1;
  lock_release(i);
}
```

4. Dynamic Home Migration

The home-based protocol has a weakness when a home node is allocated for pages that are not accessed or are less frequently accessed by the home node compared with other nodes. The wrong home allocation causes unnecessary data traffic. There is less data traffic if home nodes are chosen according to the application's memory access pattern. The strength of the home-based protocol comes from the so-called "home effect" that pages owned by a home node are always up-to-date, thus a home node needs no data communication when accessing its pages. Previously home-based protocol DSM implementations used static home allocation [17] or initial home allocation [8]. For static home allocation, a good home allocation would be decided by the application programmer. In case of initial home allocation, it cannot guarantee continuous good home allocation if an application's memory access pattern changes between barriers such as for a migratory application. Therefore dynamic home migration is needed for the home-based protocol.

To solve the home allocation problem, several home migration techniques have been proposed [4, 5, 3, 2, 15]. All home migration techniques proposed provide a solution only for single writer DSM applications or are speculative when home migration decisions are made, hoping that the future memory access pattern will be same as before.

We propose a novel home migration technique which can decide optimum home nodes for multiple writer applications as well as single writer applications. Our home migration decision is not based on a speculative assumption but is based on statistical data of the memory access pattern between two consecutive barriers, such as the sizes of diffs that need to be transferred to corresponding home nodes.

Though previous migration techniques also exploit the memory access pattern, their home migration decisions are

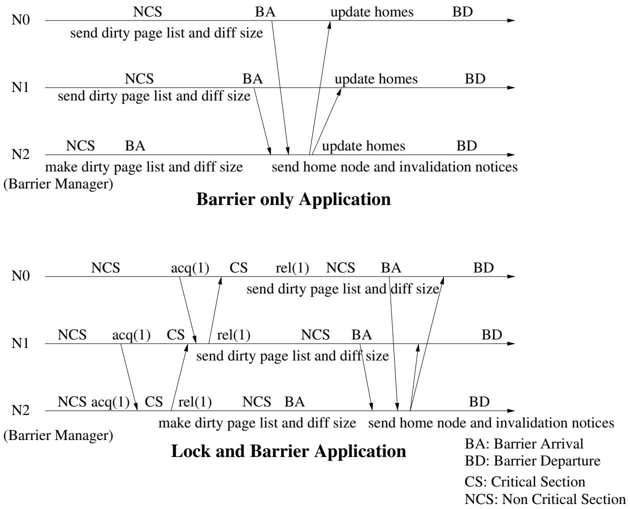

Figure 4. Home Migration Protocol

made after updating the home nodes and hope that the same memory access pattern will occur after the migration. Since their migration decision is based on the speculative assumption, effectiveness of the decisions is not guaranteed especially for those applications which have a migratory or unpredictable memory access pattern. On the other hand, our home migration decision is made before updating home nodes and is efficient since it calculates and chooses a home node which produces the minimum data traffic for home node updates.

Another direction of previous home migration techniques is to migrate homes at the time of lock synchronization. However this direction is very conservative when deciding a home migration so that many DSM applications can not fully exploit the benefit. Also, it would add more home-finding overheads [3, 2]. On the other hand, our protocol uses a barrier process in which best home nodes are piggybacked with other coherence-related data, thus minimizing the home finding and data communication overheads.

To understand our technique, we first describe the concepts of a non-critical section (NCS) diff and a critical section (CS) diff. An NCS diff is a diff made in a NCS and a CS diff is made in a CS. For barrier-only applications, only NCS diffs exist, and for lock and barrier applications, NCS and CS diffs coexist, as illustrated in Fig. 4. All nodes record their dirty pages between two consecutive barriers. Upon arrival at a barrier, all nodes create final NCS diffs. All nodes except the barrier manager node send their invalidation notices including each dirty page diff size to the manager node. Each dirty page diff size is calculated by adding the sizes of one (in case of single writer applications) or multiple NCS diffs (in case of multiple writer applications) and the last CS diffs made from the dirty page, if any. The

Application	Problem Size, Iterations	Sequential Execution Time (secs)
PNN	44,000, 235	614.10
Barnes-Hut	64k Bodies, 3	79.33
IS	2^{24}x2^{15}, 20	72.07
Water	512 mols., 10	33.07
SOR	4000x4000, 50	50.47

Table 1. Problem Sizes and Sequential Execution Times

barrier manager receives a barrier arrival notice including a dirty page list and the size of each dirty page diff from every node. Whenever the manager receives the notice, it accumulates dirty pages, creates global dirty pages, and sets a home node which has the maximum diff size for each dirty page. The idea of choosing the maximum diff size is to minimize data traffic for the home node update which should be done according to the home-based protocol. If we set the home node which has the maximum diff size, it definitely minimizes home update data traffic since the chosen home node which has the maximum diff size is not required to update the page for others and other non-home nodes update the home with smaller diffs.

We mentioned that only the last lock owner adds each CS diff size to the diff size of the corresponding dirty page. The idea is that the last lock owner will have the up-to-date values of its CS diffs made inside the critical section. Therefore, the diffs stored in the last lock owner can represent all the writes occurring inside the critical section by all nodes between two consecutive barriers. For example in Fig. 4, if N0 is the last lock owner, then only NO adds its CS diff size into the corresponding NCS diff size if any and sends them to the barrier manager. After the manager decides optimum home nodes for the dirty pages, it piggybacks the optimum home node list to all nodes as a normal barrier departure procedure. Upon receiving the new home node list, all nodes update home nodes by sending their diffs to corresponding homes. Note that only the last lock owner updates the home nodes with its integrated diffs made in the lock synchronization if the last lock owner is not the home of the CS diffs. For example, in Fig 4 CS diffs made in N1 and N2 need not to be sent to the home since they are not the last lock owner of lock 1.

The added overhead of the home migration technique is that even a home node creates twins and diffs for its home pages which diminishes one of the home-based protocol advantages. Also a barrier will take slightly longer since the home migration decision is made during the barrier time and another global barrier departure procedure is required. The performance gains due to the home migration would be dependent on the memory access pattern of an application. If the memory access pattern is regular and coarse, the benefit of the home migration will increase overall performance which overcomes the cost of the technique overhead. We will evaluate the pros and cons of the migration protocol in Section 5.3.

5. Performance Evaluation

We experimented with five DSM benchmark applications to evaluate our new protocol and techniques. We chose the following applications which are from the TreadMarks application suite except PNN. Their sequential times and problem sizes are presented in Table 1. Note that we did not change any application code in order to compare the different protocols.

- **PNN** is a parallel neural network application [14]. It trains the shuttle data set with a two pass algorithm: forward propagation and backward propagation. The data set is divided and allocated to each processor to be trained in parallel. Each node trains the network on its part of the training set and changes obtained from nodes are combined one at a time using lock synchronization. Finally node 0 applies the changes and calculates the final error. If the error is over the specified value, the training continues. The lock synchronization execution shows a migratory memory access pattern because it constantly updates the data array of changes by all nodes one at a time.
- **Barnes-Hut** is a simulation of gravitational forces using the Barnes-Hut N-Body algorithm. This application is known as an irregular fine-grained shared memory benchmark. Our Barnes-Hut implementation has only barrier synchronization.
- **Integer Sort (IS)** ranks numbers represented as an array of keys by using a counting or bucket sort. IS is known to have a migratory shared memory access pattern. IS uses only barrier synchronization.
- **Water** simulates water molecular dynamics. In each step, it calculates intra- and inter-molecular forces and potentials in a system of water molecules in the liquid state.
- **Successive Over-Relaxation (SOR)** solves partial differential equations. The shared matrix is divided into N blocks of rows on which N nodes work. The memory access pattern is very regular and exclusive so only the boundary rows between nodes require data communication.

The purpose of performance evaluation is to determine how our new protocol and techniques affect DSM performance. Previously, performance comparisons between LRC and ScC [7], and between homeless and home-based protocols [17, 16] have been presented so in this paper we do not include an in-depth analysis and comparisons between them. In Table 2, we present overall performance results of three different protocols: TreadMarks which is a homeless LRC (TM), our conventional home-based LRC (CHBLRC) and our lazy home-based ScC (LHBScC). Note that with the exception of Water, our chosen applications have no benefit by adopting ScC since three of them (Barnes-Hut, IS and SOR) do not use lock primitives, and PNN shows no false sharing. Therefore any performance improvements should have nothing to do with ScC for those applications. We chose TreadMarks as the homeless LRC implementation since it is regarded as the state of the art DSM implementation. We ran the applications over our network which has 32 nodes and is connected by 100 Mbit switched Ethernet. Each node has a 350 MHz Pentium II CPU and 192 MB of memory. All nodes are running Gentoo Linux with gcc 3.3.2.

5.1. Overall Performance Results

As presented in Table 2, our two home-based DSM implementations outperform TreadMarks in most applications except SOR. In our previous paper [16] we showed that the homeless protocol is susceptible to a hot spot and requires garbage collection, which significantly degrades the performances of most DSM applications. The strength of the homeless protocol in SOR is due to SOR's memory access pattern which is regular, coarse and exclusive. In particular, the lazy diffing technique [10] is very effective in SOR. Between two home-based implementations, LHBScC outperforms CHBLRC in all applications except Barnes-Hut and IS. In the case of IS, LHBScC outperforms the others with 32 nodes. We will explain how our two techniques improve or decrease the performance in Sections 5.2 and 5.3.

Generally LHBScC produced the least data traffic in most of the applications we tested except SOR in which TM has the least data traffic as shown in Table 3. The least data traffic over LHBScC is due to two techniques which help the better speedups in most DSM applications. The number of messages over LHBScC shows mixed results. With lock and barrier applications such as PNN and Water, it produces a relatively low number of messages. However with barrier-only applications such as Barnes-Hut, it is similar to CHBLRC in terms of the number of messages. Note that Water over CHBLRC showed poor performance. This is due to our implementation which uses the eager home update during lock synchronization. For Water, it would improve the performance to use the lazy update which updates home nodes after the lock request from other nodes. In Table 3, Water with CHBLRC over 32 nodes has no data because we couldn't run Water with CHBLRC over 32 nodes.

Apps	4 nodes			8 nodes			16 nodes			32 nodes		
	TM	CHB	LHB	TM	CHB	LHB	TM	CHB	LHB	TM	CHB	LHB
PNN	3.8	3.8	3.8	6.3	6.6	7.2	5.7	8.8	10.1	2.5	7.1	9.0
Barnes-Hut	2.0	1.9	1.9	1.9	2.3	2.1	1.3	2.6	1.8	0.9	2.8	1.3
IS	3.3	3.6	3.5	3.7	5.8	5.8	1.9	8.0	7.4	0.9	4.0	5.5
Water	2.3	1.7	2.2	3.1	1.9	3.1	3.0	1.9	3.4	2.0		2.9
SOR	2.7	1.2	2.6	3.8	1.4	3.6	4.5	1.6	4.3	4.3	1.5	4.3

Table 2. Comparison of Speed-ups between TreadMarks (TM) and our two Home-based DSM Implementations - CHBLRC (CHB) and LHBScC (LHB)

Apps.	16 Nodes						32 Nodes					
	Number of Messages			Amount of Traffic (MB)			Number of Messages			Amount of Traffic (MB)		
	TM	CHB	LHB	TM	CHB	LHB	TM	CHB	LHB	TM	CHB	LHB
PNN	119000	159446	102591	277.5	191.2	97.5	245277	325834	204777	1034.0	404.0	190.6
B-H	2802983	413393	593447	621.7	891.8	858.4	8676306	792338	1140597	1209.4	1635.8	1595.4
IS	62086	65535	66825	604.3	118.7	79.7	196074	159739	159928	2326.6	285.7	166.1
Water	188808	280922	220915	93.5	120.6	68.8	382927		419248	248.0		134.1
SOR	45471	57031	66083	65.6	221.8	87.8	63091	110929	90255	76.0	287.6	118.5

Table 3. Comparison of Data Communication Traffic between TreadMarks (TM) and Two Home-based Protocols - CHBLRC (CHB) and LHBScC (LHB)

5.2. Diff Integration Effect

To understand how our diff integration technique improves performance, we did another implementation which is exactly the same as LHB except without the diff integration. Again, since the diff integration is useful only for applications using lock synchronization, we did not test it for barrier-only applications. As illustrated in Fig. 5, the diff integration effect is much more positive in PNN compared with Water. The reason is that the critical section in PNN is large and coarse, but in Water, it is small and fine-grained. As expected, the diff integration reduces the data traffic significantly in PNN. For example, without the diff integration, it produced 619 Mbytes of data traffic over 32 nodes, but with the diff integration, it produced 191 Mbytes.

5.3. Dynamic Home Migration Effect

The performance improvement due to the dynamic home migration can be found in SOR and IS as shown in Table 2 by comparing CHBLRC which uses the static home assignment and LHBScC which uses the dynamic home migration. Note that the improved speed-ups in the barrier-only applications have nothing to do with the ScC model, LHB or the diff integration technique. They are solely due to the dynamic home migration. In particular, SOR has the most benefit showing that the speedup is more than doubled compared with CHBLRC. As for IS, the home migration prevents rapid degradation of the performance. On the contrary, the negative effect can be found in Barnes-Hut. The negative effect on Barnes-Hut can be explained from its memory access pattern which shows false sharing involving many nodes. Even though our migration protocol chooses a home node which produces minimum data traffic, the home node chosen could be the home node for many other pages between two consecutive barriers. Then the node could cause a hot spot which would degrade overall performance. In Barnes-Hut, node 0 is often chosen as the home node of the most shared pages. To find out if node 0 would be the hot spot, we made node 0 equipped with a 1 GB network interface card and executed Barnes-Hut over the network. The result showed comparable performance with CHBLRC.

In conclusion, our home migration protocol can solve the home allocation problem in regular or migratory single writer applications such as SOR and IS. However it did not improve performance for fine-grained irregular multiple writer applications such as Barnes-Hut. The protocol overhead and the memory access pattern of Barnes-Hut weakens the home effect so that the advantage of the home migration is diminished.

6. Conclusions

The home-based protocol is simple to implement and scalable compared to the homeless protocol. In most DSM applications, the home-based protocol showed better perfor-

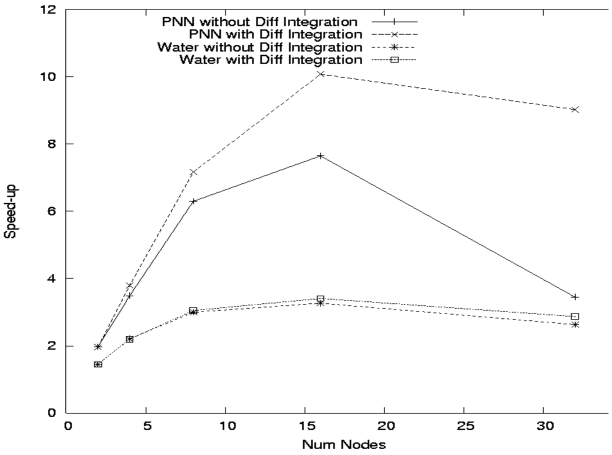

Figure 5. Diff Integration Effect on PNN and Water

mance. However the advantages of the home-based protocol depend on good home allocation. Currently static home allocation or automatic initial home allocation is used, but cannot guarantee to keep a good home allocation. Our home migration protocol can solve the home allocation problem though the benefit is reduced in fine-grained irregular multiple writer applications.

In addition to the home-based protocol advantages, we proposed the lazy home-based protocol which is more optimized by exploiting a relaxed memory consistency model such as ScC. Our novel lazy home-based protocol with diff integration will reduce the data traffic, the number of messages and the number of page faults in most lock-based DSM applications.

References

[1] 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 (SOSP-13)*, pages 152–164, 1991.

[2] B. Cheung, C. Wang, and K. Hwang. 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)*, pages 821–827, Las Vegas, Nevada, USA., 1999.

[3] J.-W. Chung, B.-H. Seong, K.-H. Park, and D. Park. Moving home-based lazy release consistency for shared virtual memory systems. In *1999 International Conference on Parallel Processing*, pages 282–290, Wakamatsu, Japan, 1999.

[4] W. Fang, C.-L. Wang, W. Zhu, and F. C. Lau. A novel adaptive home migration protocol in home-based DSM. In *Proc. of the 2004 IEEE International Conference on Cluster Computing (Cluster2004)*, pages 215–224, 2004.

[5] W. Hu, W. Shi, and Z. Tang. Home migration in home-based software DSMs. In *Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99)*, 1999.

[6] W. Hu, W. Shi, and Z. Tang. JIAJIA: An SVM system based on a new cache coherence protocol. In *Proc. of the High-Performance Computing and Networking Europe 1999 (HPCN'99)*, pages 463–472, 1999.

[7] L. Iftode, J. P. Singh, and K. Li. 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)*, pages 277–287, 1996.

[8] P. Keleher. Update protocols and iterative scientific applications. In *Proc. of the First Merged Symp. IPPS/SPDP 1998)*, pages 675–681, 1998.

[9] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92)*, pages 13–21, 1992.

[10] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, 1994.

[11] S.-K. Lee, H.-C. Yun, and J. Lee. Design and implementation of scope consistency protocol for software distributed shared memory system. In *Proceedings of the KISS Conference 2000*, pages 618–620, 2000.

[12] H. Lu, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. Message passing versus distributed shared memory on networks of workstations. In *Proceedings of Supercomputing '95*, 1995.

[13] W. E. Speight and J. K. Bennett. Brazos: A third generation DSM system. In *Proc. of the USENIX Windows NT Workshop*, 1997.

[14] P. Werstein, M. Pethick, and Z. Huang. A performance comparison of DSM, PVM, and MPI. In *Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 476–482, SW Jiaotong University, Chengdu, China, 2003.

[15] L. Whately, R. Pinto, M. Rangarajan, L. Iftode, R. Bianchini, and C. L. Amorim. Adaptive techniques for home-based software DSMs. In *13th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2001)*, pages 164–171, Pirenpolis, Gois, Brazil, September 2001.

[16] B.-H. Yu, Z. Huang, S. Cranefield, and M. Purvis. Homeless and home-based lazy release consistency protocols on distributed shared memory. In *Proceedings of the 27th conference on Australasian computer science*, pages 117–123. Australian Computer Society, Inc., 2004.

[17] 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, 1996.