Limited Multiple Writer: An Approach to False Sharing in Software DSMs*

Xianghui Xie and Chengde Han

Institute of Computing Technology, Chinese academy of Sciences
E-mail: {xhxie, hcd}@water.chpc.ict.ac.cn

Abstract

False sharing is one of the most important factors impacting the performance of DSM system. Single writer approach is simple but it can not avoid the ping-pong effect of the data page thrashing and multiple writer approach is effective to false sharing but the cost is high. This paper proposes a new approach to multiple writers in software DSM called *limited multiple writer* (LMW). It distinguishes two kinds of multiple writers with lock-based form and barrier-based form, and handles them with different policies. It combines the best aspects of both single writer and multiple writer, discards the *Twin* and *Diff* in traditional multiple writer ways, and simplifies the implementation of multiple writer in software DSM systems. The implementation of LMW in CVM software DSM system, which is based on a network of workstations, is introduced in this paper. Evaluating results show that for some applications such as SOR, LU, FFT, and IS, LMW provides a significant reduction in execution time (11%, 16%, 33% and 46%) compared to traditional multiple writer approach on our platform.

1 Introduction

False sharing is a dominant factor to the performance of software DSM systems. Generally, it occurs as more than two processors simultaneously access different parts of a same data page. There are three forms of false sharing, write-write (WW) false sharing, write-read (WR) false sharing and read-read (RR) false sharing. WW form is a key in the DSM system, especially when the consistency unit is large. It happens when more than two processors simultaneously modify different parts of a shared page. Up to now, it has been considered that the negative effects of false sharing can be reduced, but not entirely eliminated[8].

Both single writer (SW)[6] and multiple writer (MW)[11] protocols have been used to implement LRC in software DSM. SW protocols allow only a single writable copy of a page at any given time, and always transfer an entire page to satisfy an access miss. SW protocol is simple, but critically suffers from the ping-pong effect in the case of WW false sharing. The normal way for WW false sharing is that the owner maintains a minimum quantum of time when a write race occurs. MW protocols may let several

* This work is supported in part by the National Climbing Program of Chain.

writable copies of a page co-exist, and use twin-diff approach to replace entire page transferring for modifications made to a page.

MW protocols and twin-diff approach firstly emerge in Munin system[10]. Up to now, almost all-famous software DSM systems employ twin-diff approach to implement MW protocols, such as TreadMarks[3] and CVM[5], etc. With this method, twinning and diffing detect the modification. A page is initially write-protected, so that, at the first write, a protecting violation occurs. The system then makes a copy (twin) of the page, and removes the write-protection so further writes to the page can occur without any software intervention. The twin and the current copy are later compared to create a diff; a runlength encoded record of the modifications to the page. The diff is transmitted in response to requests from faulting processors.

There are some advantages using twin-diff approach in multiple writer protocols. Firstly, the correctness is secure even through most write operations are transparent to remote access mechanisms except the first write to a faulting page. Secondly, every writer only maintains its twin and diff, as the twin and diff is distributed in system, and the calculation of diffs is parallel in every node. However, certain disadvantages can hardly be avoided in twin-diff approach. One is the overhead in space. Twin and diff have to occupy certain space. The size of a twin is fixed, similar to a page size, but the size of a diff can vary from three bytes to nearly one and a half page if every element uses three bytes. Another is the overhead in time. Besides creating twin for a copy of data, every diff has to be encoded and decoded when other synchronization events occurs. This obviously spends the computing resource of related nodes, and delays the running of the nodes' processes.

In our test environment (8 SUN SPARCstation-10s connected by a 10Mbps Ethernet), we found that many applications have better performance in SW protocol than in MW protocol. The work of Keleher[6] also shows this phenomenon. There are three reasons at least. First, there hardly is write-write false sharing in many programs, especially in coarse write granularity applications. The five of eight of the benchmarks in our study do not have false sharing in LMW at all. Second, the overhead in MW protocols is high. Many single write operations have to spend extra space and time because of the simple MW protocol. The complexity of remote accesses is increased due to creating twins, encoding and decoding diffs. Finally, the overhead of communication is comparable with the overhead of the protocol computing on certain condition.

In this paper we introduce a new approach, *limited multiple writer* (LMW). It is based on the understanding to the program forms of multiple writers. Generally, there are two kinds of form of multiple writers in a program: lock-based protected operation and barrier-based synchronization operation. LMW approach distinguishes the two kinds of form and applies different policies to deal with the write-write false sharing, meanwhile, discards all twinning and diffing and merges single writer and multiple writer protocols in a same run. Evaluating results show that for some applications such as SOR, LU, FFT, and IS, LMW provides a significant reduction in execution time (11%, 16%, 33% and 46%) compared to traditional multiple writer approach on our platform.

The remainder of this paper is organized as follows. Section 2 discusses the false sharing in software DSM system, reviews the single writer and multiple writer protocol and their approaches to false sharing. Section 3 introduces the limited multiple writer approach. Section 4 describes the implementation of LMW in CVM. Section 5 shows the

test results on different protocols. Section 6 overviews the related work. Finally, Section 7 provides our conclusions.

2 Background

In this section, we briefly review the false sharing phenomenon, basic algorithm, lazy release consistency, single writer and multiple writer protocols in software DSM.

2.1 False sharing in software DSMs

In shared memory hierarchies, sharing can be classified into true sharing and false sharing[1]. The sharing of the same memory word by different processors is called true sharing, while more than two processors access different memory words that happen to be in the same block (a cache line or a page) is called false sharing. The discussion of false sharing in this paper concerns page-based software DSM systems that use lazy release consistency (LRC) and multiple writer or single writer protocols.

Generally, multi-variables (or sub-elements) store in a same multi-word block would lead to false sharing. While more than two processors simultaneously modify the different words in the same block (write-write form), the onset of false sharing will impact the performance of DSM systems. In page-based software DSM systems false sharing can have two harmful effects: extra overheads in space and time because of the coarse granularity of coherent unit, the virtual memory mechanism, the computing time for creating accurate modification information and the extra communication for memory consistency.

2.2 Basic algorithms and lazy release consistency

There are two basic forms of remote data movement in page-based software DSM systems: migration and replication. Based on the two forms, some basic algorithms was developed, such as single-reader/single writer (SRSW), multiple-reader/single writer (MRSW) and multiple-reader/multiple writer (MRMW) algorithms[2][12]. Around the basic DSM algorithms, the number of writers at the same time suffers the system performance. Under LRC protocol, the WW false sharing is the key of impacting system performance.

Lazy release consistency (LRC)[4] bases on release consistency (RC)[13] but further delays the propagation of modifications to a processor until the next relevant acquire. In addition, only the associated modifications to the chain of preceding critical sections need to be propagated on the acquire. The central intuition of LRC is that competing accesses to shared locations in correct programs will be separated by synchronization. By deferring coherence operations until synchronization is acquired, consistency information can be piggybacked on existing synchronization messages. In this way, the amount of data exchanged is minimized, while the number of message is also reduced by combining modifications with lock acquires in one message. And one of the appealing aspects of LRC is that it avoids any ping-pong effect duo to WR false sharing. If one processor writes on one part of a page and another processor reads from another part of the same page, there need not be any communication between the two processors until they subsequently synchronize. LRC protocols delay consistency actions longer than other

protocols, and therefore are more successful at hiding the effects of false sharing as well. LRC was extensively applied in some software DSM systems, such as TreadMarks, CVM, etc.

2.3 Single writer vs. Multiple writer

Single writer protocols allow only one writer for a page at any given time. Only the owner can modify the shared page. The ownership of shared pages is changed from the last writer to write requester with the page migrated.

The great advantage of single writer protocol is simple in implementation. On a write fault, the faulting processor requests ownership of the page. A static ownership algorithm is used to locate the owner. This method involves forwarding of requests through a statically assigned home processor. Ownership and the page contents are then sent from current owner to the requester. On a read fault, there is no transfer of ownership. The requester always gets a copy of the page from the owner with which it has synchronized. In either case, read fault or write fault, whole pages are sent, without any other overhead in space and time like twinning and diffing.

Unfortunately, simplicity comes at the expense of message traffic. To against WW false sharing, single writer protocols guaranty a processor a minimum quantum of time with any newly retrieved page before it can invalidated by anther processor like Mirage[14]. In most cases, this approach is effective because the WW false sharing is much less common than RW false sharing, but it can not avoid the ping-pong effect in root at all.

For reducing the effect of WW false sharing, multiple writer protocols are developed. It allows multiple writers to concurrently modify their local copy of a shared page, so that it can effectively avoid the thrashing of the page. These concurrent modifications are merged using at the next synchronization events in accordance with the definition of RC. Because software DSM levels can not accurately detect every write inside of a page, multiple writer protocols have to spend extra space and time for maintaining the coherence of the multiple copies. Since Munin implemented the multiple writer protocol with twinning and diffing in DUQ (delay update queue)[10]. Twin-diff approach has been applied to many DSM systems. In CVM, for instance, a page is initially write-protected, so that at the first write a protection violation occurs. CVM then makes a copy of the fault page, called twin, and removes the write protection so further writes to the page can occur without any software intervention. After the modification, the dirty page and the twin page are compared to create a diff; a runlength encoded record of the modifications to the page. These diffs are transmitted in response to requests from faulting processors.

The advantages of multiple writer are clear: the effects of false sharing are minimized due to processors modify their independent parts of a same page in local copies without direct communicating with other processors. The disadvantages are also clear: protocols are more complex, extra space (more than twice of the fault pages) and extra time are high.

2.4 Comparisons

We do an experiment on different protocols for obtaining the comparisons. Our experimental environment consists of eight SUN SPARCstation-10s connected by a

10mbps Ethernet. Solaris2.4 operating system and CVM (Coherent Virtual Machine) are used as platform. The applications used in this study include Water, LU, FFT, Ocean, SOR, TSP, IS, EP and MAT (reference Section 5). The protocols used in the test are MW-LRC and SW-LRC supported by CVM. The characteristics of applications are listed in Table 1, and the execution time of the applications is listed in Table 2.

Six of nine applications in the test show that SW performs better than MW except MAT, TSP and Water. This result has several root causes. First, the WW false sharing is much less common than other forms of sharing. Second, the write operation in a writable copy protected by synchronization is small; the write operations in multiple writers protected by lock present a sequential order (not concurrent order) due to small write granularity. So the minimum quantum approach can play well. Third, the overhead of SW is less than that of MW, because MW uses twin-diff approach without distinguishing single writer or multiple writers. Other three applications, TSP, Water and MAT show that MW protocol is better than SW protocol because of less communication among processors. To date, most DSM systems only apply SW or MW in their implementation. The different performance in SW and MW inspires us combining both of them to inherit the simplicity of SW and effectiveness of MW to WW false sharing. Meanwhile, reduce the overhead of MW by discarding twinning and diffing approach.

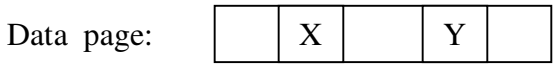
3 Limited multiple writer

Before introducing the new approach, let's consider the basic forms of shared data in programs. In the week consistency model, the accesses to shared data are always associated with synchronization events. According to synchronization, the accesses can be classified into two kinds of basic form: lock-based access and barrier-based access.

3.1 Two program forms of multiple writers

There are two forms of multiple writers in programs that can lead to write-write false sharing. They are lock-based protected operation and barrier-based synchronization operation as follows:

Figure 1. **Two forms of multiple writers in programs.**



Processor 1.	Processor 2.
barrier();	barrier();
X = 1;	Y = 2;
barrier();	barrier();
a = X;	
b = Y;	

(a). barrier-based

Processor 1.	Processor 2.
lock(1);	lock(2);
X = 1;	Y = 2;
a = X;	unlock(2);
unlock(1);	
lock(2);	
b = Y;	
unlock(2);	

(b). lock-based

Consider the case where two processors share a page. In Figure 1, variable X and variable Y stay in the same page. In (a), after former synchronization (barrier) passed, variable X and Y are concurrently modified without protection by the two processors. The all modifications are merged till the next barrier and in turn the processor 1 can read both of them correctly. In (b), variable X is related with lock 1 and variable 2 is related with lock 2. Two processors protect their accesses to variable X and Y respectively.

Studying the applications in the test, we found that the access to shared data is few in lock-based WW false sharing. It means that the granularity of protected operation in most cases is small. If we still use the twin-diff approach, the overhead in multiple writer protocols may be higher than that in single writer protocols with entire page transmission due to twinning and diffing.

3.2 Limited multiple writer

Based on the above understanding to the false sharing in software DSM systems, we notice that the two program forms may be distinguished and be treaded with different ways. So, a new approach called *limited multiple writer* to solve the WW false sharing is proposed. The principle of the approach here is, based on SW protocol, the multiple writer is allowed to happen in *limited form* and all dirty copies are collected by the owner of multiple writer till the end of *synchronization duration*. Here, *synchronization duration* is the time between two successive barriers. The *limited form* means only the barrier-based multiple writer operation is permissive and the lock-based multiple writer operations are transformed into single writer operations. Either single writer or multiple writer directly gets a writable copy from the owner and directly modifies the copy without creating a twin. The all dirty copies (entire page) are directly collected into the owner of multiple writer and compared with the original page (has saved in the owner when the MW happens) to create a new correct page without encoding diff and decoding diff. And the activity of collection is generally postponed to the end of barriers.

The LMW approach want to combine the best aspects of both single writer and multiple writer, and simplify the implementation of multiple writer in software DSM systems, meanwhile, reduce the overhead of protocol by discarding the *Twin* and *Diff* in traditional multiple writer protocol. In this way, two synchronization events are distinguished, but it dose not increase the extra burden of programming because it still follows the agreement of LRC protocols. In same synchronization duration the relationship of shared data with synchronization variables are fixed one by one, while the relationship can vary among different duration.

4 Implementation of LMW

4.1 Foundation

The Coherent Virtual Machine (CVM)[5] system is a software DSM system that supports multiple protocols and consistency models. It is written entirely as a user-level library, runs on most UNIX-like systems and was created specifically as a platform for protocol experimentation. The organization of shared memory is COMA-like. CVM included three consistency protocols: SW-SC is an implementation of page-based, single writer

protocol. It is in agreement with sequential consistency model and page ownership is migrated to each processor as a copy of the page is requested, regardless of whether the request was for a write or a read copy. SW-LRC is in agreement with lazy release consistency model and differs from SW-SC in that a single owner can co-exist with multiple readers. Pages only need to be invalidated when a processor receives a write notice via synchronization. MW-LRC is also based on lazy release consistency model, but it allows multi-processors concurrently modify the different parts of the same page.

Our LMW is based on lazy release consistency model too. Lock-based protected access is implemented by SW approach like normal single write operations in SW-LRC. Barrier-based WW false sharing leads to multiple writers modifying the same page, but the dirty copies are collected only when all processors pass the barrier. The key is distinguishing the two kinds of multiple writer dynamically. For this we use a lock-link flag piggybacked on every remote access message. If an access request follows a lock operation, the flag will be set and piggybacked on the access request to send to the data owner. If all locks have been released, following access request will piggyback a clean flag to the data owner.

4.2 Basic cases

Under the classification of multiple writers, the cases of remote access in LMW approach to a same page are simplified. We illustrate the principle with six basic cases as follows. A page has four states in the DSM level: invalidate (INV), read-only (RO), single write/read (S-WR) and multiple write/read (M-WR).

- Case 1, the page is in RO state. No mater it is or is not protected by lock when the owner received a write request, the page will be migrated to the requester at once and invalidated (into INV) by the old owner itself soon.

- Case 2, where the page is in S-WR state without lock protection when the owner received an access request without lock-link flag. On write request, the owner will send a writable copy to the requester and save a current version in the local memory, which prepares for the collection of all dirty copies soon afterwards. The state of the page will become to M-WR. Read request will lead the owner to send a current readable copy to the requester and the state is changed to RO or with other method that maintains the S-WR state.

- Case 3, where the page is in M-WR state. This is similar to the case 2 except that the owner doses not save the original version of the page when the write request arrived. On read request, the owner does not change the state, just send a readable copy to the requester.

- Case 4, where the page is protected by a lock in S-WR state. A write request causes the page migrating to the requester. A read request causing a replication of the page is send to the requester and the state is changed to RO or with other method that maintains the S-WR state.

- Case 5, when all the processors arrive a barrier, the multiple write owners can collect

all dirty copies of the shared pages and update the original pages. After this, all the state of multiple write pages in the multiple write owners change from M-WR to single WR and other writable copies are invalidated by the owners of writable copies. Out of the duration of synchronization all the pages return back to the single write environment again, and a new duration is beginning now.

- The worst case is that a page is in M-WR state when a lock-based write request arrives. It will force the owner of multiple writers' collects the relevant write copies before the next barrier event. This case is an exception to the LMW. In fact, it dose not occur at all in the applications. We deal with it in case it happens in some exceptional conditions.

4.3 Synchronization and API

The application interface (API) of LMW is similar to that of CVM. The synchronization event in the LMW includes basic lock and barrier. According to LRC protocol, acquire and release is implemented by lock and unlock. Barrier can be considered a pair of release and acquire. Release will merges the information of all the dirty pages in the processor since last release. Acquire will get the information from the lock owner and invalidate the pages depending on the information. LMW differs from SW-LRC and MW-LRC is that it distinguishes the two synchronization events. The information of multiple write pages is only visible to all processors which have arrived the barrier, but is not passed to lock requester.

5 Experimental results

We use nine applications in this study: SOR and TSP from RICE university; Water, FFT, LU and Ocean from SPLASH and SPLASH-2 suite[7] of shared memory programs; IS and EP from the NAS benchmark suite; MAT is a matrix multiplication program written by ourselves.

Table 1 summarizes the relevant characteristics of the applications. It includes for each application, the data set size used, the prevailing write granularity and the type of synchronization event. In general, under a certain machine size, the data set size impacts the activity of false sharing. We select the input size in the experiment for making the false sharing emerge fully on the eight nodes system.

Table 1. The characteristics of applications.

Application	Data size	Write gran.	Syn.
SOR	2048 x 2048	variable	b
TSP	19 cities	fine	l
LU	1024 x 1024	coarse	b
FFT	2^{18} points	coarse	b
Water	512 mols, 5step	medium	b, l
Ocean	258 x 258	coarse	b, l
IS	2^{23}	coarse	b, l
EP	2^{24}	coarse	b, l
MAT	1027 x 1027	coarse	b

Table 2 lists out the test results on eight nodes with three protocols including execution time measured by second, the number of barriers and the number of locks, the number of barrier-based multiple writer (b-MW) and the number of lock-based multiple writer (l-MW) in LMW. The number of synchronization and the number of multiple writers are only valid for the particular input set used. Some applications such as Water, TSP and Ocean show variation in multiple writer behaviors depending on the input set size and machine size. As expected, most results of LMW are better or comparable to other protocols except for TSP and Water. Comparing with MW-LRC, SOR, FFT, LU, and IS reduce execution time by 11%, 16%, 33% and 46% respectively. LWM and SW-LRC perform comparably except for MAT and TSP in which LMW performs better because there are less page transmissions in LMW when WW false sharing happens.

Table 2. The results of test running on eight nodes.

Application	LMW-LRC	MW-LRC	SW-LRC	Barrier	Lock	b-MW/l-MW
SOR	15.78	17.78	16.03	21	0	0/0
TSP	66.99	65.25	80.90	2	1127	0/4
LU	42.25	63.78	42.62	132	0	0/0
FFT	86.33	103.27	86.44	13	0	0/0
Water	53.23	46.76	54.72	36	22034	31/20
Ocean	159.31	161.02	159.82	901	2864	35/0
IS	45.94	85.30	47.85	33	227	0/0
EP	52.53	52.67	52.66	1	13	0/0
MAT	39.79	40.31	52.50	1	0	10/0

SOR divides the shared red and block array into roughly equal size bands of rows, and assign each band to a processor. The program iterates over the matrix computing a new value for each element based on its four neighbors. Communication occurs across the boundary between bands. The synchronization event is barrier only. Since the ratio of communication-to-computation is low and there is no WW false sharing for the input size we used in this application, all protocols perform well. LMW performs better than MW-LRC by 11.24%, because there is no overhead for twinning and diffing and no extra overhead in all barriers.

TSP uses a branch-and bound algorithm to find the minimum tour that starts at a designated city, passes through every other city exactly once, and return to the original city. In the application, the synchronization event is lock only, the WW false sharing is little (occurring in lock-based form with 4 times) and the write granularity is small. MW-LRC performs better because the small size of diffs. Even though there is no barrier-based multiple writer in LMW, the many transmissions of entire page from a lot of lock operations causes the performance little poor. However, the performance of LMW is significantly better than SW due to less communication.

LU is a contiguous version in our test. Its kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix as an array of blocks. The write granularity is large and there is hardly WW false sharing in it. The synchronization events are all barriers. Because the communication-to-computation is low and the locality of data is well, LMW and SW-LRC all perform better than MW-LRC. The improvement of LMW to MW-LRC is 33.76%.

FFT solves a differential equation using 1D forward and inverse FFT's. Barriers separate the phases, with a transpose being performed to optimize the computation. Communication occurs in performing the transpose, and is of a producer-consumer nature. LMW and SW-LRC have identical performance and significantly better than MW-LRC by 16.40% with zero WW false sharing in our input size, no lock-based sharing and no overhead of diffing.

Water is a molecular dynamics simulation program. The main shared data structure is the one-dimensional array of molecules allocated contiguously and partitioned among processors. The numbers of WW false sharing caused with the barrier-based form and with the locked-based form are about 31 and 20 in the input size. However, the write granularity is not large enough in the program and there are many transmissions of entire page due to a lot of lock (22034) operations. So the performance of LMW is poorer than that of MW-LRC.

IS ranks an unsorted sequence of keys using bucket sort. In this version, firstly, the keys are divided among the processors. Processors count their keys in their private buckets. Then sum up the values. The pages containing the shared buckets are completely overwritten by each processor. There is no WW false sharing and the synchronization operations used in it are barrier and lock. LMW outperforms significantly MW-LRC due to less page fault and no diffing overhead. The improvement is 46.14%.

EP is a typical "Monte-Carlo" application. Two-dimensional statistics are accumulated from a large number of Gaussian pseudo random numbers generated according to a particular scheme. It is suitable for parallel computation. All protocols perform well because of little communication, little synchronization event and no WW false sharing.

Ocean simulates large-scale ocean movements based on eddy and boundary currents. The shared data is divided by regular multi-grid. Communication in this application occurs at grid partition boundaries. The shared granularity is small and the synchronization events include lot of barriers and locks. The number of barrier-based multiple writes is about 35 and the number of lock is about 2864 in the input size. The performances of all protocols are close.

MAT is a simple matrix multiplication program with inner product algorithm. All processors with read share two arrays and with write share one array equally. For creating WW false sharing the array size is not power of two. In the program, the WW false sharing is few, but the write granularity is large. The synchronization event is just one barrier at the end of all computations. LMW performs little well than MW-LRC due to no twinning and diffing overhead and significantly better by 24.21% than SW-LRC because the write shared page is trashed between processors in SW-LRC.

In summary, LMW protocol fully inherits the best aspect of SW and partly reduces the influence of WW false sharing with barrier-based multiple writer approach. Its performance is well to the medium/coarse granularity applications such as LU, SOR, FFT, Ocean, IS, EP and MAT. But some fine granularity applications such as Water and TSP, in which the ratio of lock-to-barrier is high, do not perform well in LMW due to the high communication-to-computation. The big advantage is that LMW simplifies the implementation of multiple writer protocol. The disadvantage is that the cost of collecting dirty pages is higher in fine granularity write-write false sharing, thus, there is more demand on communication bandwidth.

6 Related work

LRC[4] is a solid foundation to our work. All protocols we implemented and used in this paper are set up on LRC. It demonstrates that the performance benefits resulting from using well protocols are considerably larger than those resulting from allowing multiple writer are. The CVM[5] system is a good experiment environment for protocols. Its modular method simplifies the implementation of other protocol approaches on it.

Many schemes to eliminate or tolerant false sharing are proposed. Most of them focus on one kind of DSM algorithm: single writer or multiple writer. The work of Amza et al is very good in protocols that adapt between single writer and multiple writer[9]. They use the ownership refusal protocol to automatically implement the transform from SW to MW and vice versa in their adaptive protocols. The MW method is similar to CVM's with twin-diff approach. They did not distinguish the different forms of multiple writers' cases and did not discard the twinning and diffing like LMW.

7 Conclusion

In this paper, we have introduced limited multiple writer approach for software DSM that deals with the two kinds of multiple writers with different algorithms. The classification of multiple writers is based on the two kinds of synchronization events, lock and barrier. Lock-based multiple writers is implemented by single writer algorithm, while barrier-based multiple writers is realized with multiple writer algorithm without twinning and diffing. Generally, all dirty writable copies are directly collected by the multiple writer owner till the end of synchronization duration. LMW approach has some characteristics as follows:

- Fully take advantage of the advantages of single writer algorithm in simplicity.
- Distinguish two kinds of multiple writer forms and deal with them with different policies.
- Simplify the implementation of multiple writer algorithm and eliminate the twinning and diffing.
- Reduce the impact of write-write false sharing to certain extent.
- Naturally transform from single writer to multiple writer and vice versa.
- Maintain the simplicity of application interface in CVM.

In our experiment, the LMW approach performs well, comparable or exceeding the performance of the traditional MW and SW protocols on seven out of nine applications. It shows that LMW is suitable for the applications with coarse write granularity. The drawback of LMW is that it need much bandwidth than traditional MW approaches' due to whole page transmission, especially, to those applications with fine write granularity. We believe that the performance of LMW will be better with the rapid progress on network technique.

Reference

[1] Josep Torrellas, Monica S. Lam, and John L. Hennessy. *Shared Data Placement Optimizations to Reduce Multiprocessor Cache Miss Rate.* In proceedings of International Conference on Parallel Processing, 1990, pages 266-270.

[2] Jelica Protic, Milo Tomasevic and Veljko Milutinovic. *Distributed Shared Memory: Concepts and Systems.* IEEE Parallel and Distributed Technology, pages 63-79, summer 1996.

[3] Peter J. Keleher, Alan L. Cox, Sandya Dwarkadas, and Willy Zwaenepoel. *TreadMarks: Distributed* Shared Memory on Standard Workstations and Operating Systems. Proceedings of the USENIX Winter 1994 Conference, 1994.

[4] Peter J. Keleher, Alan L. Cox, and Willy Zwaenepoel. *Lazy Release Consistency for Software Distributed Shared Memory.* Proceedings of the 19th Annual International Symposium on Computer Architecture, 1992.

[5] Peter J. Keleher. *The Coherent Virtual Machine.* Technical Report Maryland TR93-215, Department of Computer science, University of Maryland, Sept. 1995.

[6] Peter J. Keleher. *The Relative Importance of Concurrent Writers and Weak Consistency Models*. Proceedings of the 16th ICDCS, 1996.

[7] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh and Anoop Gupta. *The SPLASH-2 Programs: Characterization and Methodological Considerations.* In Proceedings of the 22nd ISCA, pages 24-36, June 1995.

[8] Cristiana Amza, Alan Cox, Karthick Rajamani, and Willy Zwaenepoel. Tradeoffs between False sharing and Aggregation in Software Distributed Shared Memory. ppopp 1997.

[9] Cristiana Amza, Alan Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. *Software DSM protocols that Adapt between Single Writer and Multiple Writer.* In Proceedings of HPCA'97.

[10] John B.Carter, John K.Bennett, and Willy Zwaenepoel. *Implementation and Performance of Munin.* In Proceedings of the 13th ACM SOSP, pages 152-164, October 1991.

[11] John B.Carter, John K.Bennett, and Willy Zwaenepoel. *Techniques for reducing consistency-related information in distributed shared memory systems.* ACM Transactions on Comput Systems, 13(3): page 205-243, August 1995.

[12] Michael Stumm and Songnian Zhou. *Algorithms Implementing Distributed Shared Memory.* IEEE Computer, May 1990.

[13] K. Gharachorloo, et al. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In proceedings of the 17th Annual ISCA, 1990.

[14] B. Fleisch and G. Popek. *Mirage: A coherent distributed shared memory design.* In proceedings of the 12th ACM symposium on Operating Systems Principles, pages 211-223, December 1989.