Adjusting Single-/Multiple-writer to False Sharing in Software DSMs*

Xianghui Xie and Chengde Han
Institute of Computing Technology
Chinese Academy of Sciences, P.R.China

Abstract *Two approaches are employed to deal with false sharing in software DSMs. Although single-writer is simple, it causes the ping-pong effect of the data page thrashing. In other hand, multiple-writer approach, which is effective in reduing the amount of page transfer, suffers from the high cost. This paper proposes a new approach to handle multiple writers in software DSMs called limited multiple-writer (LMW). It automatically distinguishes two kinds of multiple-writer with lock-based and barrier-based form, and deals with them with different policies. Twinning and diffing in traditional multiple-writer approach are likely to be discarded and the implementation of multiple-writer is simplified in this way. The implementation of LMW in CVM 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 approaches on our platform.*

Keywords: Memory consistency, false sharing, single-writer, multiple-writer, multiple protocols.

1 Introduction

Both single-writer [8] and multiple-writer[4] approaches have been used to implement LRC[6] in software DSM systems. Single-writer approach allows only one writable copy of a page at any given time, and always transfers an entire page to satisfy an access miss. The ragular method for write-write false sharing is that the owner maintains a minimum quantum of time when a write race occurs.It suffers from the ping-pong effect in this case although it is simple. Multiple-writer approach allows several writable copies of a page to coexist, and uses twin-diff approach to replace entire page transferring for modifications to a page[3]. Up to now, almost all-famous software DSM systems employ this method to implement multiple-writer protocols, such as TreadMarks[5] and CVM[7], etc.

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, and the calculation of diffs is parallel in every node. However, certain shortcomings are inevitable in twin-diff approach. The first is the overhead in space which is occupied by twin and diff. The size of a twin is fixed, which is similar to a page size, and the size of a diff may vary from few 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 page, every diff has to be encoded and decoded when other synchronization event occurs. This obviously occupies the computing resource of relevant nodes and delays the nodes' processes.

In this paper we propose limited multiple-writer (LMW), which is based on the konwledge of the program forms of multiple-writer: lock-based and barrier-based opera-

*This work is supported in part by Natinal Natural Science Foundation of China under grant no. 69896250 and by the National Climbing Program of Chain.

tions. LMW approach automatically distinguishes the two forms and applies different policies to deal with them. Lock-based accesses use single-write and obey LRC; while barrier-based accesses may use multiple-writer and obey RC. Meanwhile, LMW is likely to discard twinning and diffing and merges page migration and replication in a same run. Evaluating results show that some applications, such as SOR, LU, FFT, and IS, are significantly reduced in execution time (11%, 16%, 33% and 46%) in LMW, and the rest of applications are comparable in performance with traditional multiple-writer approach on our platform.

The remainder of this paper is organized as follows. Section 2 presents the limited multiple-writer approach. Section 3 describes the implementation of LMW in CVM. Section 4 shows the test results on different protocols.

2 Limited multiple-writer

2.1 Two program forms of multiple writers

In weak consistency model, accesses to shared data are always associated with synchronization events. According to synchronization, accesses can be classified into two basic forms: lock-based and barrier-based. Therefor, there are two forms of multiple writers in programs that can lead to write-write false sharing, lock-based protected operation and barrier-based synchronization operation, as shown in Figure 1.

Consider the case where two processors share a page, in Figure 1, variable X and variable Y stay in the same page, and a and b are local variables. In (a), after the former synchronization (barrier) passed, variable X and Y are concurrently modified by both processors (p0, p1) without protection. The all modifications are not merged until the next barrier and in turn the p1 can read both of them correctly. In (b), variable X relates to lock 1 and variable Y relates to lock 2. Both processors respectively protect their accesses to variable

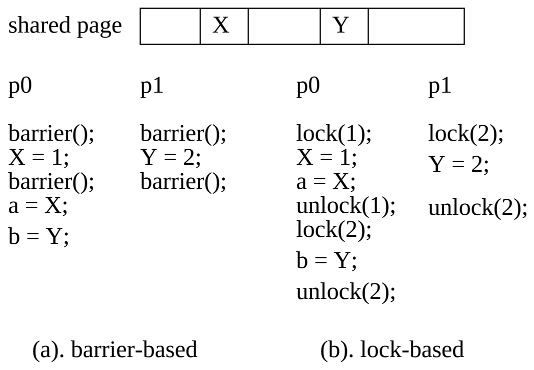

Figure 1: Two forms of multiple-writer.

X and Y.

It is not difficult to found that lock-based access to shared data is few in many applications, and 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 of single-writer protocols with entire page transmission due to twinning and diffing.

2.2 Limited multiple-writer

The two multiple-writer forms may be distinguished and treaded with different ways. The lock-based multiple-writer is treaded with single-writer method and only the barrier-based one is treaded with multiple-writer method. It is called limited multiple-writer. In this way, either single-writer or multiple-writer directly gets a writable copy of pages from the owner and directly modifies the copy in local memory without creating a twin. The owner of multiple-writer does not collect all dirty copies until the end of synchronization duration, which is the time between two successive barriers. All collected dirty copies of a page are compared with the original page (saved in the owner when the multiple-writer occurs) to create a new correct page without encoding and decoding diff.

In LMW, single-writer obeys LRC, while multiple-writer obeys RC. Comparing with LRC, the property of eagerly propagating modifications in RC is not bad because the processor does not need to maintain the writenotice and the modified data locally. If the processor learns the relevant write copies, the transmission of modifications will be effective and simple. Fortunately, in LMW, the owner of multiple-writer knows every write copy and every copy of multiple-writer knows its owner. Therefore the collection of modifications can be transmitted accurately and effectively. As long as a barrier passed, a requester can directly get the updated page from the owner.

3 Implementation of LMW

As a prototype of test, LMW is implemented in the Coherent Virtual Machine (CVM)[7] system running on a network of workstations (8 SUN SPARCstation-10s connected by a 10Mbps Ethernet).

3.1 Foundation

CVM is written entirely as a user-level library for most UNIX-like systems and created specifically as a platform for protocol experimentation. It supports multiple consistency protocols including SW-SC (single-writer with sequential consistency protocol), SW-LRC(single-writer with lazy release consistency protocol) and MW-LRC (multiple-writer with lazy release consistency protocol).

LMW is based on LRC in single-writer operations and on RC in multiple-writer operations. Lock-based protected access is all implemented by single-writer approach like SW-LRC. Barrier-based write-write false sharing causes multiple writers to modify the same page, and the dirty copies are collected only when all processors pass the barrier. It means that consistency operations are eagerly when barrier operations occur. It is critical here to distinguish the two kinds of multiple-writer dynamically. To achieve 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 requests will piggyback a clean flag to the data owner.

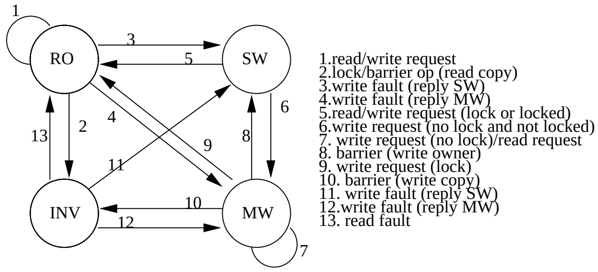

Figure 2: Page states diagram for LMW.

The collection is realized by round comparing. Every element of the original page is ready to compare to one of every dirty copy. So long as the value is not equal, the new value of the element is saved into the new page and the comparison is broke, and then another element is compared. Obviously, it differs from MW-LRC where every dirty copy compares to the twin with all elements to create diff in every copy's node, and the nodes have to maintain the diffs till garbage collection.

3.2 Basic states

Under the classification of multiple-writer, the cases of remote access in LMW approach to a same page are simplified. A shared page can be in one of four basic states: invalidate (INV), read-only (RO), single-writer (SW) and multiple-writer (MW). Each node maintains its own view of the state of the pages. Changes in the state of a local shared page are determined by the events the node observes about the page. The state transitions and their events are illustrated in Figure 2.

Initially, all shared pages in owners are in the RO state, while pages in the other nodes are in the INV state. In INV, a read fault leads the page to RO. A write fault either leads the page to SW when the owner replies SW or to MW (write copy) when the owner replies MW.

In RO, a read or write request does not change the state. A write fault causes the page into SW when the owner replies SW or into MW (write copy) when the owner replies MW.

Either write request or write fault changes the ownership whatever it is protected by lock or not. Lock and barrier operations will lead the copies of the page to INV.

In SW, a read request leads the page to RO and causes a replication of the page to be sent to the requester. A write request either with lock-link flag or the requested page has been protected by lock will lead the page to RO and migration. If the requested page is not protected by lock, a write request without lock-link flag will lead the page to MW and save a current version in the local memory, which prepares for the collection of all dirty copies soon afterwards. The owner becomes a multiple-writer owner.

In MW, a write request without lock-link flag increases the number of write copies, but does not change the state and the ownership. A read requests does not change the state, but lead to sending a current readable copy from the owner to the requester. A barrier operation will cause the multiple-write owner to collect all the relevant write copies and update the shared page. Then the page of the owner is changed to SW and all write copies become INV respectively. There is an exception when a write request with lock-link flag arrives. At this time, all write copies have to be collected immediately, and then the updated page has to be migrated to the requester and the state becomes RO. Obviously, the cost of the access is high. In fact, the worst case never occurs in the applications we used in the study.

3.3 Synchronization and API

The application interface (API) of LMW is similar to that of CVM. There is no extra burden of programming because the approach still follows the agreement of RC protocols. According to RC, acquires and releases are implemented by locks and unlocks. A barrier can be considered as a pair of release and acquire. Departure from the barrier is acquire and arrival is release. A release will merge the information of all the dirty pages in the processor since last release. An 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 in that it distinguishes the two synchronization events. The information of multiple-write pages is only seen by all processors arrived the barrier, but it is not passed to lock requester.

Table 1: The characteristics of applications.

App.	Data size	write gra.	S-time(s)	Syn.
SOR	2048^2	variable	88.14	b
TSP	19cities	fine	361.61	l
LU	1024^2	coarse	123.41	b
FFT	256K	course	14	b
Water	512,i5	medium	130	b,l
Ocean	258^2	course	26.69	b,l
IS	2^{23}	course	60.8	b,l
EP	2^{24}	course	321.29	b,l
MAT	1027^2	course	264	b

4 Experimental results

Nine applications are investigated in the study, SOR and TSP from RICE university; Water, FFT, LU and Ocean from SPLASH-2 suite[10] ; IS and EP from the NAS benchmark suite.

Table 1 summarizes the relevant characteristics of the applications. It includes, for each application, the data set size used, the prevailing write granularity, the sequential execution time in second and the type of synchronization event.We select the input size in the experiment for making the false sharing emerge fully on the eight-node system.

Table 2 lists the test results on eight nodes with three protocols including execution time measured by second, the number of barriers and locks, the number of barrier-based multiple-writer (b-MW), and lock-based multiple-writer (l-MW) in LMW (not all kinds of false sharing). 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

Table 2: The results of test on eight nodes.

App.	LMW	MW-lrc	SW-lrc	Bar.	Lock	b/l-MW
SOR	23.73	26.71	27.63	21	0	0/0
TSP	65.50	65.25	80.90	2	1127	0/4
LU	41.82	63.78	42.62	132	0	0/0
FFT	86.13	103.27	86.44	13	0	0/0
Water	47.23	46.76	54.72	36	22034	31/20
Ocean	159.01	162.82	159.82	901	2864	35/0
IS	45.94	85.30	49.23	33	227	0/0
EP	52.53	52.67	52.66	1	13	0/0
MAT	39.79	40.31	52.50	1	0	6/0

Table 3: The number of messages.

	Num.of messages		Num.of page operations.			
			LMW	MW-lrc		
App.	LMW	MW-lrc	getp	getp	twin	ap-diff
SOR	5875	4100	5671	3634	4344	325
TSP	9860	9373	7880	703	3864	26698
LU	11921	14874	2793	1509	1259	3283
FFT	31535	34572	31496	17975	14398	20250
Water	30968	28693	8672	385	3546	14920
Ocean	74771	75832	62154	14816	59378	76722
IS	7350	5648	7040	293	3397	11432
EP	55	58	23	7	8	56
MAT	9950	9952	9926	9029	1038	903

are less page transmissions when write-write false sharing occurs.

Table 3 presents the number of messages and page operations, which include getting page (getp), creating twin (twin) and appling diff (ap-diff), involved in each of our applications under MW-LRC and LMW. Although MW-LRC has few getting pages, the twinning and diffing occupy much time. LMW achieves little reduction of the number of messages in some applications and without the overhead of twinning and diffing.

SOR divides the shared red and block array into roughly equal size bands of rows, and assigns each band to a processor. All of the synchronization events employ barriers. Since the ratio of communication-to-computation is low and there is no write-write false sharing for the input size, all protocols perform well. LMW outperforms MW-LRC by 11.16%.

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 returns to the original city. All of the synchronization events are locks, the write-write false sharing is few (occurring 4 times in lock-based form) and the write granularity is small. MW-LRC performs better because the small size of diffs. However, the performance of LMW is better than SW due to fewer page faults.

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 hardly is write-write false sharing in it. The synchronization events are all barriers. Because of the low ratio of communication-to-computation and good locality of data, both of LMW and SW-LRC perform better than MW-LRC. In LMW, the number of messages is reduced significantly andthe cost of number of page operations is lower than that of MW-LRC. The improvement of LMW to MW-LRC is by 33.76%.

FFT solves a differential equation using one dimension forward and inverse FFTs. Barriers separate the phases, with a transpose being performed to optimize the computation. LMW and SW-LRC both outperform MW-LRC by 16.6% duo to no write-write false sharing, no lock-based sharing and no overhead of twinning and diffing even though no speedup at all. In LMW, the number of messages is reduced significantly and the cost of page operations is lower than that of MW-LRC.

Water is a molecular dynamics simulation program. The numbers of write-write false sharing caused by both barrier-based form and locked-based form are about 31 and 20 in the input size. However, because the write granularity is not large enough in the program, there are many page transmissions caused by a lot of lock (22034) operations and the number of message is more than that of MW-LRC. So the performance of LMW is not as good as that of MW-LRC, but it is better than that of SW-LRC.

Ocean simulates large-scale ocean movements based on eddy and boundary currents. The shared granularity is small and the synchronization events include lot of barriers and locks. The number of barrier-based multiple-write is about 35 and the number of lock is about 2864 in the input size. Both of the number of messages and page operations are less than that of MW-LRC. The performance of LMW is a little batter than that of MW-LRC even though no speedup at all.

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

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 a little communication, a few synchronization events and no write-write false sharing.

MAT is a simple matrix multiplication program. For creating write-write false sharing the size of array is not power of two. The write-write false sharing is few, but the write granularity is large. The synchronization event is just one barrier at the end of all computations. LMW outperforms MW-LRC a little due to no overhead of twinning and diffing, but significantly outperforms SW-LRC by 24.21% owing to no trashing page at all.

In summary, LMW protocol fully inherits the best aspect of SW and partly reduces the influence of write-write false sharing within barrier-based multiple-writer operations. 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 enough in LMW. The major advantage is that LMW simplifies the implementation of multiple-writer protocol. The disadvantage is that the costs of collecting and migrating dirty pages are higher in fine granularity write-write false sharing, thus, there is more demand on the bandwidth of networks.

5 Related work

Many schemes to eliminate or tolerant false sharing using multiple protocols in software DSMs are proposed. Most of them do not dynamically support multiple consistency protocols and focus on either single- or multiple-writer method. Munin[3] is the first system to propose the use of multiple protocols for different shared data objects. However, it leaves the choice of protocols to the user.

The work by Amza et al on Adaptive TreadMarks improves the performance well by adapting the protocols between single- and multiple-writer[1][2]. It uses the ownership refusal protocol to automatically implement the transform from single-writer to multiple-writer and vice versa in their adaptive protocols. The multiple-writer method uses the TreadMarks twinning and diffing mechanism, while the single-writer is similar to that of CVM. It does not distinguish the different forms of multiple-writer and does not discard the twinning and diffing like LMW.

ADSM[9] is most similar to the work by Amaz et al. It relies on a dynamic categorization technique of shared pages (called Sharing Pattern Categorization, SPC) for adapting between single and multiple-writer mode and between invalidate and update-based coherency. SPC classifies pages as falsely-shared, migratory, or producer/consumer(s). Producer/consumer(s) and Migratory pages are managed in single-writer, while falsely-shared data are managed in multiple-writer and treated under the twinning and diffing mechanism as TreadMarks. It also does not distinguish two forms of multiple-writer and differs with LMW in determining single-writer and handling multiple- writer.

6 Conclusion

We introduce the limited multiple-writer approach for software DSMs that deals with the two kinds of multiple-writer with different approaches and consistency protocols. The classification of multiple-writer is based on the two kinds of synchronization events, lock and barrier. Lock-based multiple-writer is implemented in single-writer approach, which obeys LRC, while barrier-based multiple-writer is implemented in multiple-writer approach without twinning and diffing and obeys RC. LMW approach has following characteristics:

- Fully taking advantage of the advantages of single-writer approach in simplicity.
- Distinguishing two forms of multiple writer and handling them with different policies.
- Simplifying the implementation of multiple-writer approach through multiple consistency protocols and possibly discarding the twinning and diffing.
- Reducing the impact of write-write false sharing to certain extent.
- Naturally transforming from single-writer to multiple-writer and vice versa.
- Maintaining the simplicity of application interface in CVM.

In our experiment, the LMW approach performs well, comparable or exceeding the performance of the traditional multiple-write and single-write methods 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 more bandwidth than traditional MW methods' due to whole page transmission, especially, to those applications that have fine-grained writes.

References

[1] Cristiana Amza, Alan Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. *Software DSM protocols that Adapt between Single Writer and Multiple Writer.* Proceedings of HPCA'97, 261-271.

[2] Cristiana Amza, Alan Cox, Karthick Rajamani, and Willy Zwaenepoel. *Tradeoffs between False sharing and Aggregation in Software Distributed Shared Memory.* Proceedings of PPOPP, 90-99, 1997.

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

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

[5] 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 Conference, 115-132, 1994.

[6] Peter J. Keleher, Alan L. Cox, and Willy Zwaenepoel. *Lazy Release Consistency for Software Distributed Shared Memory.* Proceedings of the 19th Annual ISCA, 13-21, 1992.

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

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

[9] L. R. Monnerat and R. Bianchini. *Efficiently Adapting to sharing Patterns in Software DSMs.* Proceedings of HPCA98.

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