Write Detection in Home-Based Software DSMs*

Weiwu Hu, Weisong Shi, and Zhimin Tang

Institute of Computing Technology
Chinese Academy of Sciences, Beijing 100080
{hww,wsshi,tang}@water.chpc.ict.ac.cn

Abstract. Write detection is essential in multiple writer protocols to identify writes to shared pages so that these writes can be correctly propagated. This paper studies different write detection schemes in a home-based software DSM system called JIAJIA. It compares the performance of three write detection schemes: the traditional virtual memory page fault write detection scheme which write-protects both home and cached pages at the beginning of an interval, a cache only write detection scheme which does not detect writes to home pages but invalidates all cached pages at the beginning of an interval, and an API write detection scheme which requires the programmer or pre-complier to explicitly records writes in program. Evaluation with some well-known DSM benchmarks reveals that tradeoffs of different write detection schemes vary with data (home) distribution and memory reference patterns of applications.

1 Introduction

Most recent software DSM systems employ the multiple writer cache coherence protocol alleviate the impact of false sharing caused by large granularity of coherence in software DSM systems. The multiple writer protocol needs to detect writes to shared memory so that the protocol can be activated to correctly propagate the writes. Write detection of home-based software DSMs is more complex than that in homeless software DSMs because detecting of writes to home pages has to be taken into account as well. This paper studies different write detection schemes with a software DSM system called JIAJIA[2]. These write detection schemes include: (1) The traditional virtual memory write detection (VM-WD) scheme which identifies writes to shared pages through page faults and twins. With this scheme, both home pages and cached pages are write-protected at the beginning of an interval to detect writes to shared pages in that interval. (2) A cache only write detection (CO-WD) scheme which does not detect writes to home pages. Only writes to cached pages are detected to generate *diff*s which are sent to their home at the end of an interval. This scheme does not write-protect home pages but invalidates all cached pages at the beginning of an interval previously. (3) An API write detection (API-WD) scheme which requires the

* The work of this paper is supported partly by National Climbing Program of China and National Natural Science Foundation of China (Grant No. 69703002).

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 909–913, 1999.

programmer or pre-complier to explicitly records writes in program. Instead of detecting writes by the software DSM system automatically, this scheme provides a function `jia_wtnt()` in the API to record writes to the shared locations.

Evaluation with some widely accepted DSM benchmarks indicate that, the tradeoffs of different write detection schemes vary with data distribution and memory reference patterns of applications, performance can be significantly improved if proper write detection is adopted in accordance with the data distribution and memory reference pattern of the application.

2 The JIAJIA Software DSM System

JIAJIA characterizes itself with a new lock-based cache coherence protocol and novel memory organization scheme which combines the physical memories of multiple workstations to form a large shared address space. Like many research or commercial systems such as TreadMarks[4], JIAJIA is implemented entirely as a user-level library and currently runs on many mainstream Unix platforms and Windows NT platform. One important characteristic of JIAJIA is its supporting of home migration scheme, which is the first home-based software DSM system that implements this idea. Multiple writer technique is employed to alleviate false sharing.

3 Write Detection in JIAJIA

3.1 Virtual Memory Write Detection (VM-WD)

With the VM-WD scheme, both home and cached shared pages are initially write-protected at the beginning of an interval. A SIGSEGV signal is delivered when a processor first writes to a shared page in the interval.

For a write fault on a cached page, a *twin* of the page is created and a write notice is recorded for this page in the SIGSEGV handler. Write protection on the shared page is then removed so that further writes to this page can occur without page faults. At the ending of the interval, a word-by-word comparison is performed between the written page and its twin to produce *diff* about this page. Write notices and *diff*s are then sent to the associated lock and home respectively.

If the SIGSEGV signal is caused by a write fault on a read-only page, then a write notice is recorded for this page and write protection on the shared page is removed. At the ending of the interval, write notices about home pages are sent to the associated lock too.

3.2 Cache Only Write Detection (CO-WD)

The above VM-WD scheme detects writes through page faults and entails additional runtime overhead on the protocol. Our previous experiments with JIAJIA

shows that, write-protecting home pages causes significant overheads for applications with large shared data set and good data distribution so that most writes hit in the home.

The CO-WD scheme reduces the overhead of home pages write detection at the cost of some extra cache miss. In the CO-WD scheme, all cached shared pages are conservatively assumed to be obsolete and are invalidated at the beginning of an interval. No write detection about home pages is required in CO-WD because the purpose of detecting write notices of home pages is to maintain coherence through invalidating associated cached pages, and the CO-WD scheme has already invalidated all cached pages when starting an interval. Only writes to cached pages are detected to generate *diff*s which are sent to their home at the end of an interval. *Diff*s are generated through comparing the dirty page with its twin as in the VM-WD scheme.

3.3 API Write Detection (API-WD)

The API write detection scheme is similar to the *dirtybit* approach[7] except that it depends on the programmer or pre-complier instead of the complier to record writes.

In the API-WD scheme, each node of JIAJIA maintains a dirty bit for each home page and a dirty bit vector for each cached page. Setting the dirty bit of a home page indicates that the page is modified, while setting the ith bit of a cached page's dirty bit vector indicates that the ith words of the page is modified. A function `jia_wtnt(addr,len)` is provided to record writes to the shared locations from `addr` to `addr+len`. The recorded shared region `[addr,addr+len)` can across page boundary. The programmer (or a pre-complier) bears the responsibility of inserting a `jia_wtnt(&a,len)` after every write to shared variable `a`. The dirty bit of each home page and the dirty bit vector of each cached page is reset at the beginning of an interval. At the ending of an interval, the dirty bits and dirty bit vectors are checked to determine whether a page is modified and which part of a cached page is modified in the interval.

4 Performance Evaluation and Analysis

The evaluation is done in the Dawning-1000A parallel machine developed by the National Center of Intelligent Computing Systems. The machine has eight nodes each with a 200MHz PowerPC 604 processor and 256MB memory. These nodes are connected through a 100Mbps switch Ethernet.

We port some widely accepted DSM benchmarks to evaluate the effect of home migration in JIAJIA. This paper shows the results of seven applications, include LU from SPLASH2[6], EP and IS from NAS Parallel Benchmarks[1], and TSP and SOR from Rice University[5].

Table 1 shows characteristics and execution results of the benchmarks. It can been seen from Table 1 that, both CO-WD and API-WD outperform VM-WD significantly in LU and SOR, while VM-WD slightly outperforms CO-WD and

Table 1. Characteristics and Execution Results of Benchmarks

Appl.	Size	Shared Mem	Seq. Time	8-proc. Time			SEGV #			Remote Accesses		
				VM	CO	API	VM	CO	API	VM	CO	API
LU	2048	32MB	84.86	25.11	24.64	24.92	32663	12072	12072	18135	20032	18135
EP	2^{24}	4KB	49.69	6.25	6.27	6.26	22	21	21	14	14	14
IS	2^{24}	4KB	30.10	4.79	4.76	4.70	230	210	210	140	140	140
SOR	2048	16MB	6.97	2.14	1.20	1.13	41200	280	280	280	280	280
TSP	19 cities	788KB	258.33	37.48	38.06	38.71	5880	6179	5711	4733	5218	4775

API-WD in TSP. The difference among three write detection schemes is trivial in EP and IS.

In LU and SOR, matrices are distributed across processors in a way that each processor only writes to its home part of the matrices in the computing. Since the computation of an iteration is synchronized with barriers and passing a barrier causes all shared pages to be write-protected in VM-WD, page faults occur for writing all home pages in an iteration. The CO-WD and the API-WD scheme, on the other hand, does not write protect shared pages on a barrier, and writing to home pages of a processor can process smoothly without any intervention. Table 1 shows that the CO-WD and API-WD scheme causes much less page faults than the VM-WD scheme in LU and SOR.

In TSP, the number of shared pages is not large and all shared pages are allocated at host 0. As a result, the advantage of CO-WD and API-WD over VM-WD in keeping home pages writable on a synchronization point is not significant. Since CO-WD invalidate all cached pages on an acquire, it has largest number of page faults among three write detection schemes. On the other hand, TSP is an application with the tight sharing memory reference pattern, a cached page is usually invalidated by the coherence protocol on an acquire because it has been written by other processors, the disadvantage of VM-WD over API-WD in write-protecting writable cached pages on synchronization points is also not significant, while the extra overhead of executing `jia_wtnt()` in API-WD dominates and VM-WD slightly outperforms API-WD.

5 Conclusions and Future Work

It can be seen from the above evaluation and analysis that write detection constitutes a significant overhead for home-based software DSMs. Evaluation results show that the tradeoffs of different write detection schemes vary with data distribution and memory reference patterns of applications, performance can be significantly improved if proper write detection is adopted in accordance with the data distribution and memory reference pattern of the application.

References

[1] D. Bailey, J. Barton, T. Lasinski, and H. Simon, "The NAS Parallel Benchmarks", *Technical Report 103863*, NASA, July 1993.

[2] W. Hu, W. Shi, and Z. Tang, "JIAJIA:An SVM System Based on A New Cache Coherence Protocol", in *Proceedings of the 7th High Performance Computing and Networking Europe*, pp. 463-272, April 1999.

[3] L. Iftode, J. Singh and K. Li, "Scope Consistency: A Bridge Between Release Consistency and Entry Consistency", in *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1996.

[4] P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel, "TreadMarks Distributed Shared Memory on Standard Workstations and Operating Systems", in *Proceedings of the 1994 Winter Usenix Conference*, pp. 115–131, January 1994.

[5] H. Lu, S. Dwarkadas, A. Cox, and W. Zwaenepoel, "Quantifying the Performance Differences Between PVM and TreadMarks", *Journal of Parallel and Distributed Computing*, Vol. 43, No. 2, pp. 65–78, June 1997.

[6] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", in *Proceedings of the 22th Annual Symposium on Computer Architecture*, pp. 24–36, 1995.

[7] M. Zekauskas, W. Sawdon, and B. Bershad, "Software Write Detection for a Distributed Shared Memory", in *Proceedings of the first International Symposium on Operating System Design and Implementation*, pp. 87–100, November 1994.