An Evaluation of the Influence of Cache-Only Write Detection on Nautilus DSM

Mario Donato Marino and Geraldo Lino de Campos*

Department of Computer Engineering - Polytechnic School - University of Sao Paulo

Abstract. Nautilus is a home-based and scope consistency DSM system. The cache-only write detection technique consists in maintaining the pages writable only on the home nodes and only detecting writes on the cache copies, a less number of page faults and page requests occur and consequently better speedups can be achieved. In this paper, the traditional write detection mechanism is compared to the cache-only write detection technique, when both are applied to Nautilus DSM. Also, in order to have a fair and homogeneous comparison, TreadMarks DSM system was included in this study. The benchmarks evaluated in this study are SOR (from Rice University), LU and Water N-Squared (both from SPLASH-2).

1 Introduction

The *Distributed Shared Memory* (DSM) paradigm[8], which has been largely discussed for the last 9 years, is an abstraction of shared memory which permits to view a network of workstations[11] as a shared memory parallel computer. By moving or replicating data[8], shared memory uniform accesses are done by the different nodes, implementing in this way the DSM main aim. These movements and/or replications of data, guarantee its consistency, allowing programs done for physically shared memory machines to be easily ported and developed[1], since to develop message passing programs is more difficult than to develop shared memory programs.

Some important DSMs like Quarks[1, 7], TreadMarks[3], CVM[10], Midway[9], JIAJIA[4] and Nautilus[5] are page-based DSM systems. Page-based solutions have achieved good speedups for several benchmarks, but there is still available place for improvements.[17]

Munin[2] DSM has proposed the multiple writer protocols technique in order to minimize the false sharing effects. The multiple writer technique allows two or more processors to write on the same page at different variables. The multiple writer protocols technique is based on diffing [1] and twinning techniques[2]. In order to do the twinning and diffing, a write detection mechanism is required.

Thus, a write detection is an essential mechanism in multiple-writer protocols to identify writes to shared pages. In order to implement multiple-writer

* {mario,geraldo}@regulus.pcs.usp.br

[1] diffs: codification of the modification suffered by a page during a critical section

protocol, software DSMs use virtual memory page faults to detect writes to shared pages. Hu[19] has proposed a cache-only write detection for JIAJIA. In this scheme, pages are protected at the beginning of an interval to detect writes in it, and also in home nodes, a write to a shared page is detected and this page will remain to be written by the home node until it is written by another node. Thus, in this interval, the page only is written by its home node and no write detection is necessary, decreasing the number of page faults and the overhead, thus improving its speedup. In addition, for applications with large shared data set and good data distribution, if the write-detection would be eliminated from the home node, a great overhead can be decreased.[16]

The cache-only write detection scheme used in Nautilus[5] is based on the scheme proposed by Hu[19] for home-based DSMs as JIAJIA[4]. And the main contribution of this paper is to evaluate the cache-only write detection scheme for Nautilus and its influence on Nautilus's speedup, verifying how this technique can help to decrease the overheads. TreadMarks[3], a reference of optimal speedups by the scientific community, is included in the comparison in order to have a reference parameter of speedups. Unhappily, the results from cache-only write-detection technique applied to TreadMarks DSM will not be showed nor compared here because the version (1.0.3) used in this study is a demo version, therefore, the source is not available.

The evaluation comparison for cache-only write-detection is done by applying three different benchmarks: LU (kernel from SPLASH-2)[15], SOR (from Rice University) and Water N-Squared (from SPLASH-2). The environment of the comparison is a 8PC's network interconnected by a fast-Ethernet shared media. The operating system used in each PC is Linux (2.x).

In section 2 a brief description of Nautilus is done. In section 3, TreadMarks is described. In section 4, write detection mechanism for Nautilus is detailed. In section 5, the environment, the applications and the results are presented. Section 6 concludes this study.

2 Nautilus

The main motivation of the new software DSM Nautilus is to develop a DSM with a simple consistency memory model, in order to provide good speedups, and also another one with a simpler user interface, totally compatible with TreadMarks and JIAJIA. This idea is very similar with the ideas utilized by JIAJIA, mentioned in the studies of Hu[4] and Eskicioglu[12], but Nautilus makes use of some other techniques, which distinguishes it from JIAJIA. These techniques will be mentioned below.

Nautilus is a page-based DSM, as TreadMarks and JIAJIA. In this scheme, pages are replicated through the several nodes of the net, allowing multiple reads and writes[8], thus improving speedups. By adopting the multiple writer protocols proposed by Carter[2], false sharing is reduced and good speedups can be achieved.

Nautilus is the first multi-threaded DSM system implemented on top of a free Unix platform that uses the scope consistency model because: 1)there are versions of TreadMarks implemented with threads, but it does not use scope consistency memory model; 2)JIAJIA is a DSM system based on scope consistency, but it is not implemented using threads; 3)CVM[10] is a multi-threaded DSM system, but uses lazy release consistency and up to the moment, it does not have a Linux based version; 4)Brazos[18] is a multi-threaded DSM and uses scope consistency, but it's implemented in Windows NT platform. The Nautilus's threads are only used to help to implement rpc services, like barriers, semaphores, page requests, while in Brazos, besides this services, threads are also used to execute user programs.

Let's summarize Nautilus features: i) scope consistency only sending consistency messages to the owner of the pages and invalidating pages in the acquire primitive; ii) multiple writer protocols; iii) multi-threaded DSM: threads to minimize the switch context; iv) no use of SIGIO signals(which notice the arrival of a network message); v) minimization of diffs creation; vi) primitives compatible with TreadMarks, Quarks and JIAJIA; vii) network of PCs and Linux 2.x; viii) UDP protocols.

Nautilus follows the lock-based protocol proposed by JIAJIA[12], because of its simplicity, thus minimizing the overheads. Figure 1 (from [12]) summarizes the state transitions. Resuming, the home nodes of the pages always contain a valid page, and the diffs corresponding to the remote cached copies of the pages are sent to the home nodes. A list with the pages to be invalidated in the node is attached to the acquire lock message.

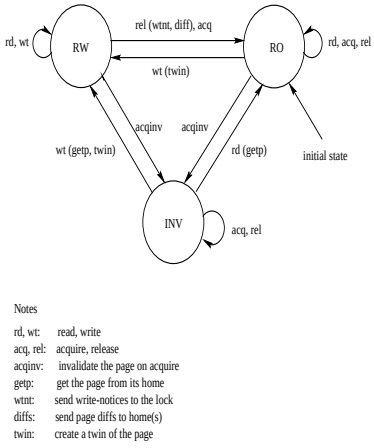


figure 1: JIAJIA Coherence Protocol [12]

In Nautilus, the owner nodes of the pages do not need to send the diffs to other nodes, according to the scope consistency model. So, diffs of pages written by the owner are not created, which is more efficient than the lazy diff creation of TreadMarks. The implementation of the state diagram of figure 1 is done in Unix with the *mprotect()* primitive, where pages can be in RO, INV or RW states, thus their states can be changed easily. As JIAJIA[4] does, Nautilus distributes

its shared pages across all nodes and each shared pages has a home node. When home nodes access their home pages, no page faults occurs. When remote pages are accessed, page faults occur, and these pages are fetched from their home node and cached locally. Instead of JIAJIA, Nautilus does not have a replacing mechanism of cached pages, since in Linux, they are replaced as memory size increases.

Nautilus uses the scope consistency memory model[14], where the coherence of cached pages is maintained through write-notices[2] kept on the lock (lock-based). As a result from the multiple-writer protocols technique application, diffs are sent to their home nodes. Briefly describing the acquire/release mechanism of Nautilus, in order to signal the end of the critical section, a release message is sent to the manager. Taking in advantage the fact of sending this message, the write notices are piggy-backed on the release message. On the acquire, the processor which is doing it sends a lock request to the manager. When granting the lock, the manager piggy-backs write-notices associated with this lock on the grant message. At the acquire, the processor, which is doing it, invalidates all cached pages that are notified as obsolete by the received write-notices. On a barrier, all write notices of all locks are cleared.

3 TreadMarks

The consistency model used by TreadMarks is the lazy release consistency[3], so the propagation of the modifications which occurred during a critical section are delayed until the next acquire. By using multiple writer protocols and the lazy release consistency model, the speedups of TreadMarks are very known, becoming it one of the most used DSM systems.

The speedups of TreadMarks made it the main DSM used by the scientific community as a reference of optimal speedups. Thus, it makes sense to compare it with other DSMs in order to have an accurate evaluation of their performance. The efficiency of TreadMarks is mainly derived from its lazy release consistency model. The major drawback of adopting this model is the high need of memory to store the diffs all over the user's application execution. Thus, the size of the benchmarks used to evaluate the speedups of the DSM system can be compromised if there is not enough memory to execute the program or if the operating system does swap. If it cannot use enough size to run the benchmarks, the relation computation versus synchronization becomes unfavorable to use a DSM system.

Let's summarize TreadMarks features: i) lazy release consistency and its variations [3], minimizing the number of consistency messages through the net; ii) multiple writer techniques of Munin [1]; iii) primitives compatible with m4; iv) IBM SP2, Sun Sparc, PCs; v) AIX, Solaris, free Unix (Linux 2.x); vi) UDP protocols to minimize network protocols overhead; vii) first DSM to have a speedup compatible to a shared memory machine[3].

[2] write-notices: indication of which pages were modified during the critical section

4 Cache-Only Write Detection

Munin[2] DSM has proposed the multiple writer protocols technique in order to minimize the false sharing effects. The multiple writer technique allows two or more processors to write on the same page at different variables. The multiple writer protocols technique is based on diffing and twinning techniques[2]. The diffing consists in codify the modifications suffered in a critical section. In order to do twinning, a write detection mechanism is needed to produce the twin of the page. Second generation page-based DSMs basically use a pair SIGSEGV signal/handler to detect the write on a page and a pair malloc() and bcopy() to create the twin. As it can be seen, a signal and several system calls are necessary to produce it.

As other DSMs like TreadMarks[3] and JIAJIA[4] do, Nautilus uses virtual memory page faults to detect writes to shared pages. Shared pages are protected at the beginning of an interval (several critical sections). When the first write to a shared page occurs, a SIGSEGV signal is delivered, and in this moment the page can be written without protection. At the end of the critical section, Nautilus sends the write-notices related about the shared pages.

Several studies [3, 4, 17, 19, 20] show that the detection of writes to shared pages presents significant overheads. Other studies show that applications with large shared data set and good data distribution, the writes hit in the home. The study of Amza[16] showed that in many applications, single-writer constitutes the dominant part of the sharing behavior and shared pages are normally written by the home node (owner) for a certain interval. Thus, it is possible to conclude that for applications with large shared data set and good data distribution, if the write-detection would be eliminated from the home node, a great overhead can be decreased. And going further, it is not necessary to become the page from the read-write state to read-only state[19, 20] (as it can be seen in figure 1) if only the home node writes to this page. Also, only in cache copies (remote nodes), writes are detected and, when this occur, the pages go from read-write to read-only state. Concluding, if the home page is written in some interval, several *mprotect()* and SIGSEGV handlers calls are saved, improving the DSM's speedup. If the home page is not written by its home in the interval, some unnecessary invalidations of remote cached pages can occur, thus more remote accesses.

Taking into account the single-writer behavior presented in several applications, Nautilus implements its cache-only write detection scheme which recognizes automatically a single write to a shared page by its home node, presuming that the page will continue to be written by its home node, until it will be written by remote nodes.

5 Experimental Platform, Applications and Result Analysis

The results reported here are collected on a 8 PC's network. Each node (PC) is equipped with a K6 - 233 MHz (AMD)processor, 64 MB of memory and a fast

Ethernet card (100 Mbits/s) . The nodes are interconnected with a hub. In order to measure the speedups, the network above was completely isolated from any other external networks. Each PC runs Linux Red Hat 6.0. The experiments are executed with no other user process.

The test suite includes three programs: LU (from SPLASH-2[15]), SOR (from Rice University) and Water N-Squared (from SPLASH-2).

The LU kernel from SPLASH-2 factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The NxN matrix is divided into an nxn array of bxb blocks (N = n*b) to exploit temporal locality on sub-matrix elements. The matrix is factored as an array of blocks, allowing blocks to be allocated contiguously and entirely in the local memory of processors that own then[4].

Water is an N-body molecular simulation program that evaluates forces and potentials in a system of water molecules in the liquid state using a brute force method with a cutoff radius. Water simulates the state of the molecules in steps. Both intra- and inter-molecular potentials are computed in each step. The most computation- and communication-intensive part of the program is the inter-molecular force computation phase, where each processor computes and updates the forces between each of its molecules and each of the n/2 following molecules in a wrap-around fashion[12].

SOR from Rice University solves partial differential equations (Laplace e-quations) with a Over-Relaxation method[4]. There are two arrays, a black and red one allocated in shared memory. Each element from red array is computed as an aritmethic mean from black array and each element from black array is computed as an aritmethic mean from red array. Communication occurs across the boundary rows on a barrier.

Before presenting the results and their analysis, it is necessary to emphasize that the execution time for number of nodes=1 in all evaluated benchmarks is obtained from the sequential version of the benchmarks without any DSM primitive. So, the primitive used to allocate memory to obtain the sequential time (number of nodes=1) is **malloc(),** default primitive of C programming. In order to have an accurate, homogeneous and fair comparison, the same programs are executed TreadMarks(version 1.0.3 demo) and Nautilus (version 0.0.1).

The data input size N used in the LU and SOR evaluation is **1792x1792**; the number of iterations for the SOR benchmark is **10** . The number of steps used in Water is **25** and the number of molecules is **1728**.

Table 1 shows some features and results of the benchmarks: sequential time (t(1)), 8-processor parallel run time(8), speedup (Sp), remote get page request counts (gp) and number of local SIGSEGV of Nautilus(SG). The sequential time t(1) was obtained from the sequential program without no DSM primitives and **malloc()** primitive. In order to evaluate the cache-only write detection speedup, remote get page request counts and the number of local SIGSEGVs of Nautilus are taken. For table 1, **Tmk** means TreadMarks, **NautV** means Nautilus with the traditional virtual memory write detection and **NautCO** means Nautilus with the cache-only write detection. NautCO assumes that a page will only be

written by its home in the future barrier interval, keeping its home page writable if only the home writes to it.

There are some constraints with TreadMarks version (1.0.3) used:

i) the applications were executed and the speedups measured using *Nautilus* running on up to **8 nodes;**

ii)**bigger input sizes:** the shared memory size is limited in this version;

iii)the source of this demo version is not available, thus it was neither possible to evaluate TreadMarks with other page size nor to measure the parameters gp (get page request) and SG (local SIGSEGV).

application	LU	Water	SOR
t(1)	350.90	2983.00	29.10
t(8).Tmk	55.45	403.20	8.66
t(8).NautV	54.32	426.40	7.66
t(8).NautCO	49.60	422.07	4.37
Sp.Tmk	6.33	7.40	3.36
Sp.NautV	6.46	7.00	3.80
Sp.NautCO	7.07	7.07	6.66
SG.NautV	7980	10210	12425
SG.NautCO	440	824	927
gp.NautV	1528	505	118
gp.NautCO	340	448	74

table 1: comparing TreadMarks and Nautilus (traditional and cache-only)

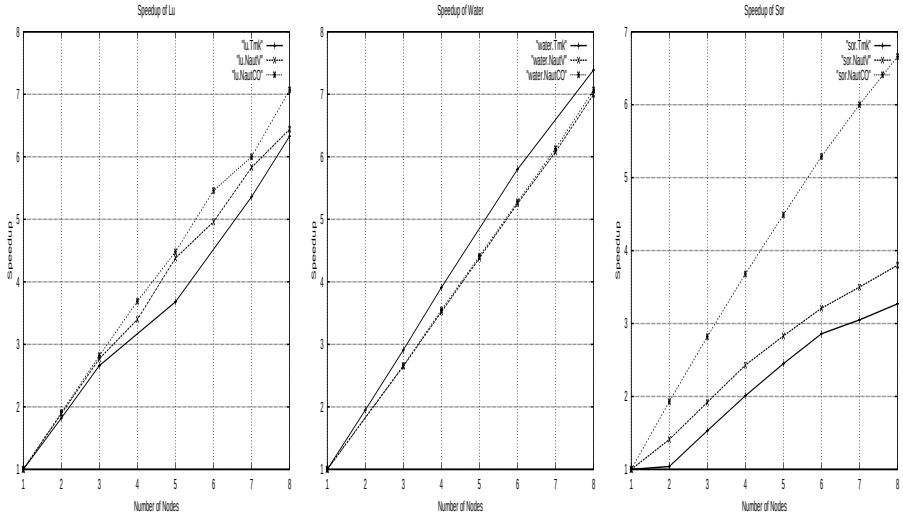
Three general conclusions can be taken from table 1: the number of SIGSEGVs and the number of page requests decreased when the cache-only write detection is applied; also the speedups increases with cache-only write detection applied. It can be seen lower remote page accesses (gp rows) in NautCO than in NautV, because of the correct home page write assumption. It was said before that if a home page is assumed to be written by its home next barrier interval and if the home does not write it, the assumption causes unnecessary invalidation of remote cached pages and consequently some other page requests. This justifies the decreasing of the number of SIGSEGVs and page requests.

Now, for each benchmark, the behavior of the cache-only technique is analyzed.

5.1 LU

By looking at figure 2, the speedups of LU can be seen. It can be noticed from this figure that the cache-only write detection technique improved Nautilus's speedup. Analyzing the cache-only write detection technique in LU, matrices are distributed across processors in a way that each processor writes to its home part of the matrices in the computing.[19, 20] Since the computation of an iteration is synchronized with barriers and passing a barrier causes all shared pages to be write-protected in traditional virtual memory, page faults occur for writing all home pages in an iteration. The cache-only method does not write protect shared pages on a barrier, and writing to home pages of a processor can process without any SIGSEGV.

From table 1, for eight nodes, it can be noticed that for Nautilus, the cache-only write method improved the speedup up to 9.44% The increasement of the speedups can be justified by observing, from table 1, the reduction of the number of SIGSEGVS and the number of page requests both by an order of magnificence lower, when comparing NautCO to NautV.



figures 2,3 and 4: speedups of LU **(N=1792)**, Water **(1728 molecules** and **25 steps**) and SOR **(1792x1792)**

Comparing with TreadMarks, for eight nodes, NautV outperforms it by 2.05% and NautCO outperforms it by 11.69%. Also, an adequate choice of the page owners (data distribution) of Nautilus improves data locality and gives a lower cold start up time to distribute shared data. In addition, the elimination of SIGIO signals minimizes the overheads of Nautilus.

5.2 Water

By looking at figure 3, the speedups of Water can be seen. It can be noticed from this figure that the cache-only write detection technique improved Nautilus's speedup.

From table 1, for eight nodes, it can be noticed that NautCO is up to 1.00% faster than NautV . This behavior can be justified by observing the number of SIGSEGVS and the number of page requests both from table 1, where the NautCO version has an order of magnificence lower SIGSEGVs than NautV and a also 11.28% lower number of page requests.

Confronting TreadMarks with NautV and NautCO, for eight nodes, TreadMarks is up to 5.71% faster than NautV and up to 4.67% faster than NautCO. This behavior can be justified because of the high synchronization, which is the dominant feature of Water and because of the Nautilus's semaphore implementation.

5.3 SOR

By looking at figure 4, the speedups of SOR can be observed. Also from the speedup's curves, the cache-only write detection technique improved Nautilus's speedup.

In SOR, as the same way in LU, matrices are distributed across processors in a way that each processor writes to its home part of the matrices in the computing.[19, 20] Since the computation of an iteration is synchronized with barriers and passing a barrier causes all shared pages to be write-protected in traditional virtual memory, page faults occur for writing all home pages in an iteration. The cache-only method does not write protect shared pages on a barrier, and writing to home pages of a process without any SIGSEGV. From table 1, for eight nodes, it can be noticed that for NautV, the cache-only write method improved the speedup up to 75.26%. The increasement of the speedups can be justified by observing the number of SIGSEGVS from table 1, an order of magnificence lower for the NautCO version compared to NautV. The number of page requests were reduced too.

When compared to TreadMarks, NautV outperforms it up to 13.10% and NautCO outperforms it up to 98.21%, the last an excellent speedup. The better choice of the page owners, the multi-threading and the elimination of SIGIO signals also help to improve Nautilus's speedup.

6 Conclusion

The contribution of this study is an evaluation of the influence of the cache-only write detection mechanism on the speedup of Nautilus DSM. Also, the speedups of TreadMarks and Nautilus DSMs were compared with three different programs.

This study shows that the cache-only write detection improved Nautilus's speedup up to 9.44% for LU application, and 75.26% for SOR benchmark, which showed that these benchmarks had shared data and single-writer behavior. For Water, the cache-only does not contribute meaningfully to increase the speedup. Other important conclusion is the reduction of one order of magnificence of number of SIGSEGVs and the number of request page faults when the studied technique was applied.

In future works, other benchmarks will be evaluated in this comparison. Also, a complete version of TreadMarks will be evaluated, in order to compare the cache-only write detection of this DSM with Nautilus.

References

1. Carter J. B., Khandekar D., Kamb L., *Distributed Shared Memory: Where We are and Where we Should Headed*, Computer Systems Laboratory, University of Utah, 1995.
2. Carter J. B., *Efficient Distributed Shared Memory Based on Multi-protocol Release Consistency*, PHD Thesis, Rice University, Houston, Texas, September, 1993.

3. Keleher P. , *Lazy Release Consistency for Distributed Shared Memory*, PHD Thesis, University of Rochester, Texas, Houston, January 1995.
4. Hu W., Shi W., Tang Z., *JIAJIA: An SVM System Based on a new Cache Coherence Protocol,* technical report no. 980001, Center of High Performance Computing , Institute of Computing Technology, Chinese Academy of Sciences, January, 1998.
5. Marino M. D.; Campos G. L.; *A Preliminary DSM Speedup Comparison: JIAJIA x Nautilus*, to be published at HPCS99, Kingston, Canada, June, 1999.
6. Li K, *Shared Virtual Memory on Loosely Coupled Multiprocessors*, PHD Thesis, Yale University, 1986.
7. Swanson M., Stoller L., Carter J., *Making Distributed Shared Memory Simple, Yet Efficient*, Computer Systems Laboratory, University of Utah, technical report , 1998.
8. Stum M. , Zhou S. , *Algorithms Implementing Distributed Shared Memory*, University of Toronto, IEEE Computer v.23 , n.5 , pp.54-64 , May 1990.
9. Bershad B. N. , Zekauskas M. J., SawDon W. A., *The Midway Distributed Shared Memory System* , COMPCOM 1993.
10. Keleher P., *The Relative Importance of Concurrent Writers and Weak Consistency Models*, in Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS-16), pp. 91-98, May 1996.
11. Becker D., Merkey P.; *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs*, Proceedings, IEEE Aerospace, 1997.
12. Eskicioglu, M.S., Marsland T.A., Hu W, Shi W.; *Evaluation of the JIAJIA DSM System on High Performance Computer Architectures*, Proceeding of the Hawai'i International Conference on System Sciences, Maui, Hawaii, January, 1999.
13. Hu W. , Shi W., Tang Z.; *A lock-based cache coherence protocol for scope consistency*, Journal of Computer Science and Technology, 13(2):97-109, March, 1998.
14. Iftode L., Singh J.P., Li K; *Scope Consistency: A bridge between release consistency and entry consistency.* Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA'96), pp. 277-287, June, 1996.
15. Woo S., Ohara M., Torrie E., Singh J.P., Gupta A.; *The SPLASH-2 programs: Characterization and methodological considerations. In Proceedings of the 22th Annual Symposium on Computer* Architecture, pages 24-36, June, 1995.
16. Amza C., Cox A. L., Dwarkadas S., Jin L. J., Rajamani K., Zwaenepoel W., *Adaptive Protocols for Software Distributed Shared Memory*, Proceedings of IEEE, Special Issue on Distributed Shared Memory, pp. 467-475, March 1999.
17. Iftode L., Singh J. P.; *Shared Virtual Memory: Progress and Challenges*; Proceedings of the IEEE, Vol 87, No. 3, March 1999, 1999.
18. Speight E., Bennett J. K., *Brazos: A third generation DSM system*, In Proceedings of the 1997 USENIX Windows/NT Workshop, pp. 95-106, August, 1997.
19. Hu W., Shi W., Tang Z., *Adaptive Write Detection in Home-based Software DSMs*, to appear in Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, August 1999, Redondo Beach, CA.
20. Hu W., Shi W., Tang Z., *Write Detection in Home-based Software DSMs*, to appear in Proceedings of Euro-Par'99, August 31-September 2, Tolouse, France.