Time Domain Performance Evaluation of Adaptive Hybrid Cache Coherence Protocols

Danko Ivosevic, Sinisa Srbljic, and Vlado Sruk

Abstract —**Adaptive hybrid cache coherence protocols use both the write-invalidate mechanism and the write-update mechanism to maintain coherence among copies of data objects. Each of these protocols implements a decision function that chooses the appropriate mechanism in order to improve their performance. In most existing solutions, decision functions are based on communication traffic. Moreover, the authors of the adaptive protocols use communication traffic as a performance measure in their papers.**

In contrast, in this paper we present the results of a performance evaluation of adaptive hybrid cache coherence protocols in both the traffic domain and in the time domain. We compare three adaptive protocols with pure write-invalidate and pure write-update protocols. Let r_{WI}, r_{WU}, and r_A be the average communication traffic per access for the write-invalidate protocol, the write-update protocol, and the adaptive protocol, respectively. The adaptive protocol *minimizes the traffic* if $r_A \approx min(r_{WI}, r_{WU})$. Similarly, the adaptive protocol *minimizes the access latency* if $t_A \approx min(t_{WI}, t_{WU})$, where t_{WI}, t_{WU}, and t_A are the access latencies for the write-invalidate protocol, the write-update protocol, and adaptive protocol, respectively. For some of the workload parameters the adaptive protocols minimize both traffic and access latency. However, we also present and analyze the workload parameters for which adaptive protocols minimize communication traffic, but fail to minimize the access latency.

Index Terms — **Adaptive hybrid cache coherence protocol, decision function, time domain performance evaluation.**

I. Introduction

In recent years, most computer systems, from tightly-coupled multiprocessor systems to loosely-coupled distributed systems, based on either local networks or global networks (Internet), use data replication and caching to improve their performance. Two factors that affect the performance of a system with replicated data are the data access behavior of the application and the choice of the cache coherence protocol. Cache coherence protocols maintain coherence among copies of the same data object. Application developers and users either choose the cache coherence protocol appropriate for the given data access behavior, or tune the application to run efficiently with the given cache coherence protocol.

D. Ivosevic, S. Srbljic, and V. Sruk are with the School of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia. E-mail: divose@yahoo.com, {sinisa, sruk}@zemris.fer.hr

Most cache coherence protocols use one of two basic coherence mechanisms: the write-invalidate mechanism or the write-update mechanism. Adaptive hybrid cache coherence protocols use both mechanisms. Each of these protocols implements a decision function that chooses the appropriate mechanism in order to improve the performance of the adaptive hybrid system. We evaluate three adaptive cache coherence protocols, called hereafter RWB adaptive protocol [1], EDWP adaptive protocol [2], and APCUM adaptive protocol [3] (Adaptive Protocol using CUMulative cost). The performance of the adaptive protocols is compared with the performance of the pure write-invalidate protocol and the pure write-update protocol. We simulate the cache coherence protocols for a network of workstations [4]. Since the decision functions of most adaptive protocols choose the mechanism that incurs the lowest communication traffic, the authors of the adaptive protocols choose traffic as the performance measure.

However, for most applications the final performance goal is not to minimize the communication traffic, but to reduce the access latency. Therefore, we decided to evaluate the adaptive protocols in both the traffic domain and the time domain. We chose the average traffic per access as the performance measure in the traffic domain and the average latency per access as the performance measure in the time domain.

The simulation model and basic coherence mechanisms are presented in Section 2. Section 3 briefly describes the adaptive cache coherence protocols. We discuss the simulation results in Section 4. Final comments on the performance evaluation are given in Section 5.

II. The Simulation Model and Basic Cache Coherence Mechanisms

We simulate a distributed system, where seventeen workstations share one data object [5]. One of the seventeen workstations, denoted as the *sequencer* [4], performs global sequencing in order to ensure coherence among the copies of a given data object. In addition, the sequencer is the home location for the given data object. All of the other sixteen workstations are called *clients*. If a client does not have a copy of the data object that it needs, it sends a request to the sequencer. Requests that are cache hits (both reads and writes) are executed *locally*, while other requests that are cache misses are called *remote*. A local access does

0-7803-6290-X/00/$10.00 ©2000 IEEE

not incur communication traffic and, in addition, we assume that the latency for a local access is equal to zero.

The seventeen workstations have a bandwidth of 10 Mb/s (megabits per second) available for communication. The workstations communicate via packets of three different sizes: the size of a *small packet* is 45 B (bytes), the size of a *medium packet* is 90 B and the size of a *large packet* is 720 B. Small, medium, and large packets transmit command messages, updating information, and data objects, respectively. Workstations exchange packets in order to maintain cache coherence. For example, if a client reads a data object from the sequencer, it takes: one small packet containing the read request sent from the client to the sequencer and one large packet containing the data object sent from the sequencer to the client. Since a broadcast sends packet to each workstation one by one, the communication traffic for the broadcast is sixteen times higher than the traffic for the basic packet transfer.

If an access is a write, the write-update protocol updates the caches associated with all workstations [4]. Therefore, after a cold start, all reads in the steady-state are cache hits, and they execute locally with no communication traffic and have zero latency. Since all writes broadcast the updating information to all sixteen workstations, writes are always remote accesses.

If an access is a write, the write-invalidate protocol updates only the local cache associated with the workstation that performs the write [4]. All other copies in the caches associated with other workstations are invalidated, by broadcasting the invalidation command message to all sixteen workstations. If there are no intervening accesses from other workstations, after the first write access all successive reads and writes from the same workstation hit to the only valid copy in the whole system. Therefore, these accesses are local, and they execute locally with no communication traffic and have zero latency. In addition, second and all successive reads from the same workstation, without intervening writes from other workstations, are also local. The first read or write, after a write from another workstation, is remote. If multiple workstations have valid copies, then the write is a remote access that broadcasts the invalidation command message to all workstations.

III. Adaptive Cache Coherence Protocols

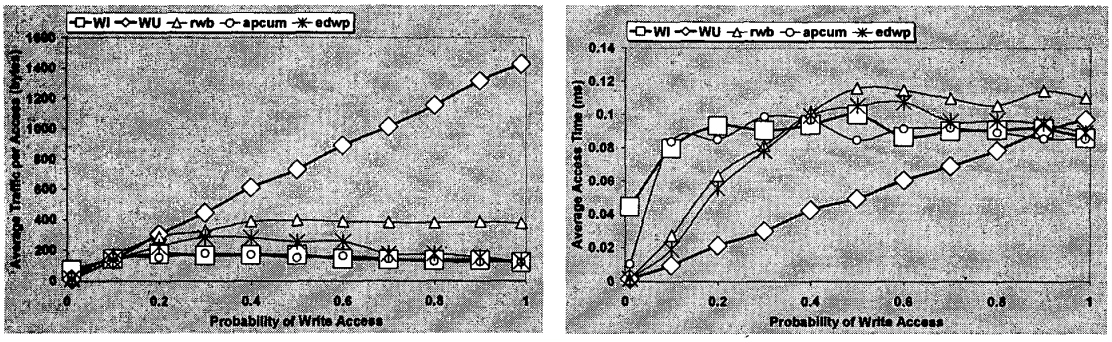
Most decision functions are based on a sequence of consecutive accesses to a data object. Copies of a data object are updated until a certain number of consecutive writes is performed by the same workstation, such that no intervening accesses are performed by other workstations [1-2], after which the write-invalidate protocol is used. For example, we simulate the RWB adaptive protocol [1] whose decision function switches from the write-update protocol to the write-invalidate protocol after three successive writes from a workstation without intervening writes from other workstations. We also simulate the EDWP adaptive protocol [2]. The difference between the RWB protocol and the EDWP protocol is that not only write accesses from other workstations, but also the read accesses, are used to prevent switching from the write-update mechanism to the write-invalidate mechanism. Both adaptive protocols switch from the write-invalidate mechanism to the write-update mechanism on the first intervening access from another workstation.

If there are no intervening accesses, then only one workstation accesses the data object. Therefore, it is better to use the write-invalidate protocol than the write-update protocol. The write-invalidate protocol sets the state of the local copy as the only valid copy on all workstations, and all successive reads and writes from the same workstation become local with zero traffic. If the write-update protocol is used, then the traffic is not zeroed and depends on the probability of write accesses, because each write access for the write-update protocol is remote.

However, if there are intervening accesses from other workstations, the write-update protocol generates lower traffic than the write-invalidate protocol. If the interleaving of accesses from different workstations increases, the traffic generated by the write-update protocol does not change, because it depends only on the probability of write accesses. In contrast, the traffic generated by the write-invalidate protocol increases. After a write from one of the workstations, all other copies are invalidated. If a workstation hits to a copy that is invalidated, the whole valid copy should be transferred from the sequencer or another client (a client can also have the only valid copy, while the sequencer's copy is invalidated). The traffic cost for transferring the copy could be high and it depends on the size of the data object. Therefore, the traffic for the write-invalidate protocol is significantly affected by the intervening accesses from other workstations.

In addition, we also simulate an adaptive protocol called APCUM, whose decision function is based on the cumulative communication traffic [3]. The decision function chooses between the write-invalidate protocol and the write-update protocol based on which mechanism incurs lower cumulative communication traffic. The decision function implements two counters for each data object. One counter keeps the cumulative traffic for the write-invalidate protocol, and another counter keeps the cumulative traffic for the write-update protocol. Although only one of the protocols (write-update or write-invalidate) is running at the time, both of these two counters are updated. To maintain the cumulative traffic, the appropriate communication costs are added to the counters for each read and write access. The APCUM protocol switches to the mechanism whose counter has a lower value.

Results of the simulation and time domain analysis presented in the next section show that the previously described adaptive protocols may minimize both the traffic and the latency for some workload parameters, but there are workload parameters for which they will fail to minimize the access latency.



(*a*) Average traffic per access

(*b*) Average access latency

Figure 1: Simulation results

IV. PERFORMANCE EVALUATION

The analysis of real applications shows that the workstations perform accesses in bursts, i.e. only one workstation accesses the data object at a time [3]. Therefore, we use Dubois & Wang burst model [6] to define the artificial workload. We define that the artificial workload generates the accesses in bursts, where the burst size (number of accesses in access burst) is determined by a Normal (Gaussian) distribution *N(mean, standard deviation)*. When a workstation completes a burst of accesses, the probabilities of performing the next access burst are the same for all seventeen workstations.

Figure 1 shows the results of simulation using the following workload parameters: the burst size is equal to *N*(11, 1). The average size of a burst, i.e. the number of accesses performed by a workstation without intervening accesses from other workstations, is equal to eleven. The standard deviation is small and equal to one, which means that in most of the access bursts there are eleven accesses. Both figures give the simulation results for different probabilities of write accesses, ranging from 0% to 100% (for example, 20% is denoted as 0.2 and 100% is denoted as 1). We assume that workstations perform on average ten accesses per second. We present the graphs for the write-invalidate protocol (WI), for write-update protocol (WU), and all three adaptive protocols (RWB, EDWP, and APCUM).

Figure 1*a* presents the average communication traffic per access in numbers of bytes. As discussed in the previous section, for large access bursts the traffic for the write-invalidate protocol is not significantly affected by the percentage of write accesses. Figure 1*a* confirms that the average traffic for the write-invalidate protocol does not change significantly when the probability of a write access is increased. Since each write is a remote access for the write-update protocol, the average traffic for the write-update protocol increases with the probability of a write access. If the probability of write accesses is greater than 10%, the average traffic for the write-update protocol becomes larger than the average traffic for the write-invalidate protocol. Since all of the simulated decision functions minimize the average traffic, the average traffic for the adaptive protocols is similar to the minimal value between the average traffic for the write-update protocol and the average traffic for the write-invalidate protocol.

Figure 1*b* presents the average access latency in milliseconds (ms). Although the average traffic for the write-update protocol is higher than the average traffic for the write-invalidate protocol for probabilities of writes between 10% and 90%, the average access latency for the write-update protocol is lower than the average access latency for the write-invalidate protocol. There are two reasons for this:

1. *The write-update protocol uses smaller packets per access than the write-invalidate protocol.* Smaller packets use network bandwidth more efficiently. The write-update protocol uses only medium and small packets, while write-invalidate protocol uses large and small packets.

2. *The latency of a remote access for the write-update protocol is lower than the latency of a remote access for the write-invalidate protocol.* To calculate the latency of a remote write access for the write-update protocol, we need to add the latency of a medium packet (updating information sent from a client to the sequencer) and the latency of a small packet (write permission command packet sent from the sequencer to a client). Different types of remote accesses for the write-invalidate protocol have different latencies. For example, if a client reads a valid copy from the sequencer, one small packet travels from the client to the sequencer (read request command message) and one large packet travels back (data object response packet). If a client reads a valid copy from another client, the latency is doubled, because the communication between two clients is performed through the sequencer: a read request message travels

from the requesting client to the sequencer (one small packet), a read request message travels from the sequencer to the client that has the only valid copy (one small packet), the data object travels from the client that has the only valid copy to the sequencer (one large packet), and the data object travels from the sequencer to the requesting client (one large packet)[1].

However, if the percentage of writes is higher than 90%, the access latency for the write-update protocol outgrows the access latency for the write-invalidate protocol. Since each write is a remote access for the write-update protocol, the access latency for the write-update protocol increases with the probability of a write access. In addition, the access latency for the write-update protocol is increased, because the increase in traffic increases the average time that the packets wait in queues in order to be sent through network.

As we can see from the Figure 1, the adaptive protocols minimize both traffic and access latency for percentages of writes that are either less than 10% or greater than 90%. For the percentages of writes less than 10%, the adaptive protocols use the write-update protocol, which incurs lower traffic and lower access latency than the write-invalidate protocol. For the percentages of writes higher than 90%, the adaptive protocols use the write-invalidate protocol, which incurs lower traffic and lower access latency than the write-update protocol.

For the percentage of writes higher than 10% and lower than 90%, the adaptive protocols minimize the traffic, but do not minimize the access latency. The adaptive protocols use the write-invalidate protocol, which incurs lower traffic than the write-update protocol. Since the write-invalidate protocol incurs higher access latency than the write-update protocol, the adaptive protocols incur high access latency. If the percentage of write accesses is between 60% and 80%, the access latency for adaptive protocols is even higher than the access latency for the pure write-invalidate protocol. During the first three accesses in a burst, the RWB and EDWP protocols use the write-update protocol. Since there are no intervening accesses from other workstations, after three successive write accesses from the same workstation the adaptive protocols switch to the write-invalidate protocol. Therefore, the first access in the next burst must read the only valid copy from another workstation. Since the first access in the burst has the same latency as the first access of the write-invalidate protocol, and since there are two successive accesses that have the same latencies as accesses for the write-update protocol, the average latency for the adaptive protocol is higher than the average latency for the pure write-invalidate protocol.

Since it is not possible for the APCUM protocol to maintain precise cumulative traffic counters, the adaptive protocol unnecessary switches between the write-invalidate protocol and the write-update protocol, which increases the average access latency.

[1] We can estimate the values for the average access latency presented on Figure 1 based on the packet service time (packet size in bits over bandwidth in bits per second). The average latency for the write access for the write-update protocol is (90B+45B)/(10Mb/s)≈0.1ms. For example, if 50% of accesses are writes, the average access latency is 0.1ms/2≈0.05ms (Note that for the write-update protocol all writes are remote and all reads are local with zero latency). Let us assume that 100% of accesses are writes. Since the first write access in a burst is remote and all other ten writes are local with zero latency (there are 11 accesses in burst), the average access latency for the write-invalidate protocol is (2×720B+2×45B)/(11×10Mb/s)≈0.1ms.

V. CONCLUSION

The results of simulation presented in this paper show that it is not possible to design an adaptive cache coherence protocol that minimizes both the average traffic per access and the average access latency. There are workload parameters for which the write-invalidate protocol minimizes the average traffic per access, while the write-update protocol minimizes the average access latency. Since the analyzed decision functions switch between the write-update protocol and the write-invalidate protocol in order to minimize the communication traffic, the access latency for the adaptive protocols may outgrow the average access latency of both pure write-invalidate protocol and pure write-update protocol.

VI. ACKNOWLEDGMENTS

The authors wish to thank A. Grbic, I.Benc, I.Crkvenac, R. Radovic, I..Skuliber, and M..Stefaneč for providing valuable comments on the research presented in this paper.

VII. REFERENCES

[1] L. Rudolph and Z. Segall, "Dynamic Decentralized Cache Scheme for MIMD Parallel Processors", In Proceedings of the 11th *Annual International Symposium on Computer Architecture*, Ann Arbor, Michigan, Pages 340-347, June 1984.

[2] J. Archibald, "A Cache Coherence approach for Large Multiprocessor Systems", In International *Conference on Supercomputing*, St. Malo, France, pages 337-345, July 1988.

[3] S. Srbljic, "An Adaptive Coherence Protocol Using Write Invalidate and Write Update Mechanisms", *CIT*, Vol. 4, No. 3, September 1996, pp. 187-197.

[4] S. Srbljic, "Modification of Cache Coherence Protocols for the Distributed Environment", *Automatika*, Vol. 34, No 1-2., pages 29-40, January-April 1993.

[5] D. Ivosevic, "Decision Functions of Adaptive Cache Coherence Protocols", Diploma Thesis, University of Zagreb, School of Electrical Engineering and Computing, Zagreb, July 1999. (Thesis published in Croatian, original title:" *Funkcije odlučivanja u samoprilagodljivim postupcima za održavanje jednoznačnosti uvišestručenih podataka*")

[6] M. Dubois and J.-C. Wang, "Shared Block Contention in a Cache Coherence Protocol", IEEE Transactions on Computers, Vol. 40, No. 5, pages 640-644, May 1991.