


Brett D. Fleisch, University of California, Riverside
Heiko Michel, University of Kaiserslautern
Sachin K. Shah, IBM Corporation
Oliver E. Theel, Darmstadt University of Technology

 To make complex computer systems more robust and fault tolerant, data must be replicated for high availability. The level of replication must be configurable to control overhead costs. Using an application suite, the authors test several distributed shared memory coherence protocols under different workloads and analyze the operation costs, fault tolerance, and configurability of each.

Fault Tolerance and Configurability in DSM Coherence Protocols

Potentially malfunctioning components in large distributed shared memory systems require highly available services that can be configured according to expected failure rates in the environment. Although several coherence protocols have been developed for DSM systems,¹ few address configurability and fault

tolerance. To address these aspects, the DSM coherence protocol must offer increased redundancy, decreased reliance on centralized data and control, support for servicing requests locally, and control over the degree of data availability on a per-data-unit basis. In a page-based DSM system, as assumed in this article, the unit of interest is a DSM page. Object-based systems can use the same protocols. The Boundary-Restricted (BR) class of coherence protocols² satisfies these properties and offers highly available access to shared data at low operation costs.

In the past, our work focused on a refinement of BR—called *Dynamic* BR (DBR)³—and investigated BR's as well as DBR's properties using analytical techniques for a *single* DSM page. In this article, we use a shared-memory simulator and a DSM application suite to further study the BR class of protocols. Our goal is to investigate the trade-offs between the degree of fault tolerance, operational costs, and configurability for various DSM coherence protocols such as Write-Invalidate, Write-Invalidate with Downgrading, Write-Broadcast, and various instances of

the BR class. We have chosen to execute real-world applications in a simulation environment. These programs generally use more than a single DSM page, and not all of these pages are equally distributed among the participating sites as execution proceeds. Thus, the behavior of a single DSM page under a certain coherence protocol might differ from the results reported here; however, our results enable a more realistic judgment of the pros and cons of a particular coherence protocol.

Impact of the Coherence Protocol

Data replication can improve performance in distributed systems by reducing the latency to access data and enabling more requests to access data concurrently. Consequently, a system can tolerate more malfunctioning components because several copies of the data exist. Hence, a direct consequence of replication is greater data availability, that is, the probability that at an arbitrary point in time a system can access the data of a DSM page. Although it can potentially improve fault tolerance, replication

also leads to a fundamental problem—the difficulty of ensuring that all replicas are mutually consistent and that the sites are not accessing out-of-date (or *stale*) data. As a result, each time a shared page is modified, consistency mechanisms must ensure that all existing copies are consistent. Page-based DSM systems usually do this by either transmitting the modified page or an invalidation message to the various sites, or by transmitting only the modified section of the page. The cost for maintaining mutual consistency in replicated systems can be very high.

Restricting the number of replicas lowers operation costs because fewer expensive updates or invalidations must be made to ensure consistency. However, fewer replicas also decreases concurrency and the level of data availability. In developing robust systems, we try to maximize data replication and minimize the costs of consistency-related message transmission. Clearly, there is a strong relation between the DSM coherence protocol, operation costs, and data availability. Some systems have addressed fault tolerance for DSM,⁴ but most do not use data replication to provide high availability. On the other hand, our work does not consider *operational availability*—the operations required to recover a system and its applications so that replica fail-over occurs and operational fault tolerance is provided. This latter topic is beyond the scope of this research.

WRITE-INVALIDATE PROTOCOL

The Write-Invalidate (WI) protocol permits multiple readers or a single writer to access a DSM page, but not both simultaneously. Typically, any number of readers can concurrently access a page that is being accessed in read mode. When a process tries to write to the page, the system multicasts an invalidation to all other sites storing the page. When a site storing the page receives the invalidation, the site discards the DSM page and acknowledges the multicast. In addition, one site transfers the latest copy of the data to the site where the write request originated. In a careful implementation of the protocol, when all sites respond to the sender, the write is permitted to complete. Consequently, processes are prevented from

reading stale data because they do not store a replica when a writer is present. The effect is to process all updates to the item sequentially, while reads can proceed in parallel.

WRITE-INVALIDATE WITH DOWNGRADING PROTOCOL

The WI Protocol with Downgrading (WID) is a modification of the WI protocol used in DSM systems such as Mirage⁵ and Mirage+.⁶ The primary difference between WI and WID occurs after a write. During the next shared read access from another site, the write copy is not invalidated as in WI. Instead, the page remains stored and readable at the former writer's site. The data from the former writer is then transferred to the new

Clearly, there is a strong relation between the DSM coherence protocol, operation costs, and data availability.

reader. Therefore, in WID, a minimum of two readers are typically using the page in read mode. Subsequently, a write to one of the pages that is in read mode might involve a mere upgrade to write mode along with an invalidation to the other site.

The WID protocol works well when a reader and a writer try to access the same page from different sites. Although a cost must be paid to transfer the data from a former writer to a new reader, the reverse is not true, because in WID a mere upgrade and invalidation can be performed when one of the two readers writes to the page. For these situations, WID can be much less expensive to use.

COMMON PROPERTIES OF WI AND WID

Both WI and WID are multiple-readers/single-writer protocols. During writes in both WI and WID, data availability is poor because only one copy of

the data is present in the network. Furthermore, the application's read-write ratio will govern the degree of availability: if it is low (that is, many writes occur compared with the number of reads), availability will be low. When control transfers from the writing process to other reading processes, the data becomes much more available. However, the read-write ratio of an application is a tenuous property on which to base availability.

From a cost perspective, WI and WID are suited for applications where the number of successive writes between two reads is high, as well as applications that exhibit a high degree of per-site locality of reference.² Additionally, there is no stable system state (when the number of copies of a page and their location remain constant), because replicas are continually being invalidated whenever a write request is executed.

WRITE-BROADCAST PROTOCOL

The Write-Broadcast (WB) coherency protocol is an update-based protocol. When a site obtains write access to a page, the process updates the page locally and multicasts the changes to all other sites possessing replicas. Sites that store replicas incorporate the updates that are received and send acknowledgments. Stored replicas are never deleted in WB. Once a site obtains a copy of the page as a result of a read or write request, the item continues to remain at the site for the duration of the program. Because every write operation requires the multicast of update (or *control*) messages for consistency, write operations are expensive,⁵ particularly when components are not failure-free and write operations have to be aborted when a site cannot be contacted due to site or communication failures. However, read operations are extremely inexpensive after the initial cost of transferring the page to the site, because reads are local and do not require remote communication or cooperation with other sites. In addition to allowing multiple readers, several processes may write the same page at the same time in WB; this is known as multiple-readers/multiple-writers sharing. Once all sites cache a copy of the data item, the system is said to have reached a stable system state.²

Table 1. Modes (two-dimensional tuples) associated with DSM pages at client sites.

MODE	DESCRIPTION	SAMPLE SITUATION
(Local read, local write)	Data can be read and modified locally.	A client has the <i>only</i> cached copy of the DSM page.
(Local read, global write)	Data can be read locally, but write requests must be submitted to the DSM server.	The client site is <i>one of many</i> active readers of the DSM page.
(Global read, global write)	Data cannot be read or written locally. Requests must be submitted to server.	The requested DSM page is not cached at the client site.
(Global read, local write)	Not used	None

Table 2. Boundary settings of different coherence protocols. N defines the maximum number of sites in the network, and R_{min} , R_{max} , W_{min} , and W_{max} are determined by the parametric settings supplied to BR.

PHASE		PROTOCOL			
		WI	WID	BR	WB
Read	Minimum copies	1	2	R_{min}	1
	Maximum copies	N	N	R_{max}	N
Write	Cached copies	Always deleted or invalidated	Always deleted or invalidated	Updated, possibly deleted	Never deleted, always updated
	Minimum copies	1	1	W_{min}	1
	Maximum copies	1	1	W_{max}	N

The BR Coherence Protocol Class

The hybrid protocol BR combines the advantages of WI and WB by using a multiple-readers/multiple-writers approach to enhance concurrency and parallelism. BR typically reduces the number of replicated copies during write operations but, unlike WI, lets more than one replica exist during writes. Therefore, BR provides greater fault tolerance in terms of data availability than WI. Unlike WB, you can specify a maximum number of replicas to control update costs, which otherwise could grow in proportion to the number of sites accessing the page.

DESIGN GOALS AND FUNCTIONAL MECHANISM

We believe that a coherence protocol for a large, error-prone DSM system must exhibit the following properties:²

1. Limited workload dependability: the number of copies of a page should have limited dependence on the workload—that is, the sequence of read and write operations.
2. Lower bound in the number of cached copies: the number of cached

copies of a DSM page preferably should never be reduced to one. Single copies make the DSM system vulnerable to component failures, since only one copy of the data exists. Nonetheless, while increasing the lower bound on the number of cached copies results in higher data availability, it also increases the management cost.

3. Upper bound in the number of cached copies: the number of cached copies should never result in greater management cost than overall benefit. The protocol should avoid situations where all clients cache replicas that must be updated during writes. Consequently, an arbitrary linear increase in the number of sites caching a page must also be avoided, since the probability that a write operation can successfully complete decreases significantly in this case.

The BR protocol, which has been designed along these three design properties, operates as follows. On receiving the requested page, a client maps the data into its memory and uses it according to the *granted* mode. The item at a client site is considered *cached* in a particular mode

with respect to a particular request. A mode is a two-dimensional tuple consisting of a read attribute in the first dimension and a write attribute in the second dimension. The mode determines how subsequent read and write operations function. Table 1 gives the various modes used by the functional model.

If the client reads (writes) the data of a page cached with a local read (local write) mode attribute, then this results in a local read operation (local write operation) carried out at the client site. If a particular page or item is cached in a mode including a global read (global write) attribute, then a read (write) request needs to be submitted to the DSM server. This request is called a global read operation (global write operation). Sequential consistency⁵ is the basis for our protocol's memory coherence policy, but weaker forms of consistency can be supported as well.

The BR protocol can be in one of two phases at a time: A so-called read phase starts with the execution of the first global read operation the DSM server receives after a global write operation and lasts until the next global write operation. A second phase, the write phase, starts with the execution of the first global write operation the DSM server receives after a global read operation and ends with the next global read operation. So, in a read phase, no global write operations are executed, whereas in a write phase, no global read operations occur. Local operations do not change the phase state of the BR protocol. A detailed description of BR and its behavioral characteristics is available elsewhere.²

Figure 1. Data availability depending on the number of cached copies and site availability.

PROTOCOL INVARIANTS

The BR coherence protocols exhibit boundary settings as shown in Table 2. The DSM server enforces these constraints while serving global read and global write requests that are sent to it. During a global read operation, BR guarantees the number of read copies of a page N_r :

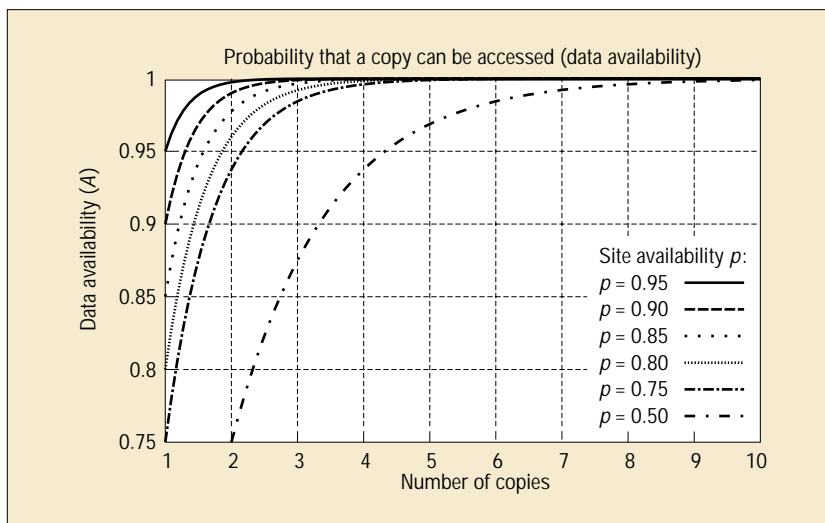
$$\begin{aligned} R_{min} \leq N_r \leq R_{max} \quad \text{with} \quad R_{min} \leq N_r, \\ R_{max} \in \{1, 2, \dots\} \end{aligned} \quad (1)$$

where R_{min} and R_{max} represent the minimum and maximum number of cached copies in a read phase. Further, BR also guarantees that during a write phase, the number of copies of a page, N_w , is

$$\begin{aligned} W_{min} \leq N_w \leq W_{max} \quad \text{with} \quad W_{min} \leq N_w, \\ W_{max} \in \{1, 2, \dots\} \end{aligned} \quad (2)$$

where W_{min} and W_{max} represent the minimum and maximum number of cached copies during a write phase.

An instance of the BR coherence protocol class is addressed as BR(R_{min} , R_{max} , W_{min} , W_{max}). Practical settings for these parameters are discussed elsewhere.² In the scope of this article, we focus on a particularly interesting subclass of BR, denoted as BR(w , n). We obtain this subclass from the general BR coherence protocol class by setting $w = W_{min} = W_{max}$, $n = R_{max}$, and $R_{min} = w + 1$. Restricting W_{min} and W_{max} in this way helps to keep costs during a write phase very low while still providing an acceptable level of availability if $w > 1$. Setting $w > 1$ avoids the undesirable situation where the number of copies drops to just one. By setting R_{min} to $w + 1$ and R_{max} to n , $w + 1$ up to n copies of a page can exist during a sequence of global read operations (n is introduced to simplify the notation). In particular, at the beginning of a read phase only a single additional copy must be installed, thereby minimizing the costs for this operation. Creating at least one additional copy at the beginning of a read phase ensures higher availability without causing excessive costs. Thus, the BR(w , n) subclass takes into consideration that maintaining a certain level of data availability during a read phase costs less than maintaining



the same level of availability during a write phase.

Note that if the parameter w is set to 1 and n to N (the number of sites in the network), then the resulting BR(1, N) protocol is equivalent to the WID protocol.

From Table 2 it is clear that the WI, WID, and WB protocols violate design property number 2 given earlier by permitting a single copy to exist during read and write phases. WB violates design property number 3 when all clients cache replicas during the write phase. On the other hand, BR protocols restrict the number of cached copies to lie between defined limits. Ideally, the boundary-setting can be different for every page, leading to a different degree of availability per item and different operation costs.

The behavior of WI, WID, WB, and BR is best understood by examining each protocol during read and write operations. Consider the first operation performed on some DSM page p . Assume it is a read operation. WI, WID, and WB will satisfy the request by sending page p to the requesting site; at this point there is only one copy of p in the network. On the other hand, BR immediately stores a replica at R_{min} different sites. Any failures that occur at the requesting site can be tolerated using BR because of the other existing replicas. This is not the case in WI, WID, or WB.

Examining the protocols during write operations, WI, WID, and WB occupy extremes of a spectrum. At the lower end is the lack of replication offered by WI and WID. At the higher end is the level of replication exhibited by WB—maximal. BR occupies the entire area and can be configured to lie

between extremes—never minimal and never maximal. It is this region between extremes that presents a suitable compromise between the desired level of replication and operation costs.

Figure 1 illustrates the dependency between site availability p and data availability $A_{data}(p) = 1 - (1 - p)^a$ with $a > 0$ being the number of copies of a particular page. If all sites of a network (or at least those sites caching a copy of a certain page) are available with a probability of, for example, $p = 0.75$, and if only a single copy of a page is present in the system, then the data availability of this page is, of course, $A_{data}(0.75) = 75\%$. If the number of copies always available is increased to $a = 2$, then the associated data availability is increased by approximately 18%. Thus, when using WI, WID, or WB, minimal data availability would be 75% in the present example whereas by using BR(2, n) with $n \leq N$, minimal data availability would be approximately 93% and, by using BR(3, n), approximately 98%. Various other examples can be derived by the given graphs. All those examples demonstrate the superiority of BR coherence protocols in terms of data availability if W_{min} and W_{max} are appropriately set.

A Comparison

Figure 2 shows the behavior of BR(2, 5) in a network of five sites with a sample reference trace. All sites request or reference the same page. Recall that BR(w , n) guarantees that w replicas exist during a write phase and that r ($w + 1 \leq r \leq n \leq N$) replicas exist during the read phase. In BR, two types of messages are

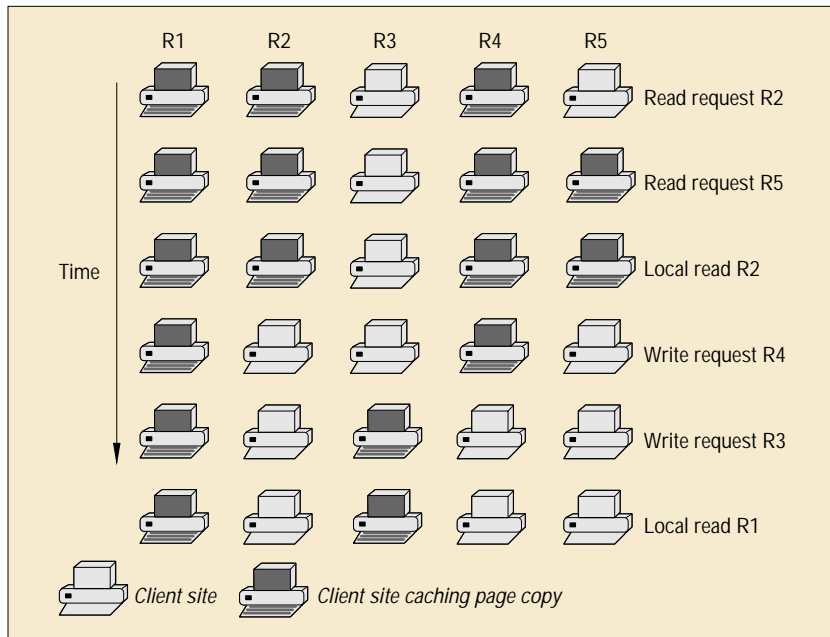


Figure 2. Location of copies of a single page for the BR(2, 5) coherence protocol. “Read Request R2,” for example, means that site R_2 requested a DSM page for (global) reading and the request has been granted.

transmitted across the network: *control* and *data* messages. Control messages contain page invalidations, page downgrades, page upgrades, or page updates. Data messages contain actual page data being transferred to a requesting client.

The example begins with the first operation, a read operation issued by R_2 , which requires R_2 to receive a single data message containing the new page. BR(2, 5) guarantees that at least three read copies are present during a read phase; hence two more page replicas must be

stored. Three replicas are shown in the first step of Figure 2. The system uses a total of three data messages to install the three replicas, as shown in Table 3. As a postcondition of this operation, a page copy at site R_2 is *compulsory* but two other copies must exist at two other sites. In our example, these sites are R_1 and R_4 . (The current implementation of BR uses a policy that randomizes where the additional replicas are stored, but to optimize performance, system designers may select other policies based on page usage statis-

tics.) Table 3 shows the total number of copies of a page existing after the execution of the corresponding operation, on a per-operation basis.

The next read request at site R_5 requires that a page be transmitted to site R_5 . The local read operation at site R_2 requires no messages. The next write request at site R_4 takes place in two phases. BR(2, 5) must maintain only two page copies during a write phase. As a result, first the system transmits two control messages to invalidate page copies at R_2 and R_5 (again, picked randomly in this example). Only then does the write operation proceed at R_4 . And second, the system transmits a single control message to site R_1 to update the replica. The next operation, a write request at site R_3 , requires that a page be sent to R_3 and an invalidation control message be sent to randomly chosen site R_4 . Finally, the system sends a single control message to update the replica at R_1 due to the write operation at R_3 . The last operation in the example, a local read at site R_1 , takes place without any message costs. Only two copies of the page exist in the final phase of the example, because the page mode is a (local read, global write); the request is

Table 3. Comparing the number of messages transmitted and the number of replicated copies available for different coherence protocols on a five-site network. In BR(3, 5), R_5 is randomly picked to cache an additional copy of the page.

OPERATION		NUMBER OF MESSAGES					TOTAL NUMBER OF COPIES				
		WI	WID	BR (2, 5)	BR (3, 5)	WB	WI	WID	BR (2, 5)	BR (3, 5)	WB
Read R_2	Control	0	0	0	0	0					
	Data	1	1	3	4	1	1	1	3	4	1
Read R_5	Control	0	0	0	0	0					
	Data	1	1	1	0	1	2	2	4	4	2
Local Read R_2	Control	0	0	0	0	0					
	Data	0	0	0	0	0	2	2	4	4	2
Write R_4	Control	2	2	3	3	2					
	Data	1	1	0	0	1	1	1	2	3	3
Write R_3	Control	1	1	2	3	3					
	Data	1	1	1	1	1	1	1	2	3	4
Read R_1	Control	0	1	1	0	0					
	Data	1	1	0	0	1	1	2	2	3	5
Breakup	Control	4	4	5	6	5					
	Data	5	5	5	5	5					
Total		9	9	10	11	10	8	9	17	21	17
Avg. per operation		1.5	1.5	1.6	1.8	1.6	1.3	1.5	2.8	3.5	2.8

Table 4. Extending the protocols over a larger network ($N = 10$ sites) and then comparing the number of messages transmitted and the number of copies available.

OPERATION	NUMBER OF MESSAGES					TOTAL NUMBER OF COPIES				
	WI	WID	BR (2, 5)	BR (3, 5)	WB	WI	WID	BR (2, 5)	BR (3, 5)	WB
Total from previous example	9	9	10	11	10	8	9	17	21	17
Read R_{10}										
Control	0	0	0	0	0					
Data	1	1	1	1	1	2	3	3	4	6
Write R_8										
Control	2	3	3	4	6					
Data	1	1	1	1	1	1	1	2	3	7
Read R_7										
Control	1	1	0	0	0					
Data	1	1	1	1	1	1	2	3	4	8
Write R_6										
Control	1	2	3	4	8					
Data	1	1	1	1	1	1	1	2	3	9
Read R_9										
Control	1	1	0	0	0					
Data	1	1	1	1	1	1	2	3	4	10
Breakup										
Control	9	11	11	14	19					
Data	10	10	10	10	10					
Total	19	21	21	24	29	14	18	30	39	57
Avg. per operation	1.7	1.9	1.9	2.2	2.6	1.3	1.6	2.7	3.5	5.2

satisfied locally, and the server is never contacted. The system does not enforce the invariant in Equation 1 because it is still in a write phase (local reads do not cause the system to enforce the read phase invariant while being in write phase).

In addition, Table 3 compares the behavior of WB, WI, WID, BR(2, 5) and BR(3, 5) for the same sequence of read and write operations as shown in Figure 2, including the number of messages transmitted and replicated copies available.

Table 4 presents a similar comparison but with a larger network of 10 sites. It extends the sample sequence of read and write operations to present a more complete analysis. According to our observations, the class of BR protocols proved to be most effective in providing the following design properties.

- *Competitive operation costs.* Table 3 shows that in terms of the total number of messages sent, BR costs as much as WI and WB. In a larger network (see Table 4), BR transmits fewer messages than WB and only a few more control messages than WI.
- *Better fault tolerance in terms of high data availability.* In the smaller network (Table 3), WB, WI, and WID permit *single* DSM pages to exist at times. In WB, sufficient replicas usually do not exist during application startup. With WI and WID, this problem persists throughout the execution of the application. On the other hand, BR never permits the number of page copies to be less than W_{min} .

- *Better scaling than WB.* Observe the behavior of WB for the *Write* R_8 and *Write* R_6 operations in Table 4. The system is approaching a stable system state; numerous replicated copies of the same page are being stored, which in turn requires numerous control messages to keep the replicas consistent. Clearly in large networks, as the system reaches a stable state, WB incurs very high operation costs for write operations, making it scale poorly. BR, on the other hand, can be set to maintain $W_{min} \leq w \leq W_{max}$ copies, thus limiting the number of messages transmitted. Consequently, BR is configurable to maintain fixed levels of operational costs even with a large increase in the number of participating sites.
- *Configurability.* Table 4 shows that WB maintains too many replicas that must be kept mutually consistent. WI and WID, on the other hand, provide too few replicas and thus cannot provide high data availability. The class of BR protocols lets us control the level of replication by varying W_{min} and W_{max} as required by the fault tolerance requirements of the system. The analytical results in Table 4 confirm this conjecture.

The Simulator and Simulator Applications

While the comparison example described earlier confirmed that BR is cost effective and provides fault-tolerant

data access, a goal of this work is to examine BR using real shared-memory programs. We created a program-driven simulator that executes commonly studied DSM programs.^{9,10} The simulator lets us compare and analyze WI, WID, WB, and BR in actual practice.

The simulations use AINT (*Alpha Interpreter*),⁷ a software tool for analysis of shared-memory systems. AINT simulates parallel programs on uniprocessor workstations and provides a program-driven simulation environment. In AINT, a simulation application is executed until it generates a memory reference. AINT then transfers control to the back-end, which simulates the desired coherence protocol in response to that memory reference event. The back-end, coded in C, enforces the coherence protocols we are studying such as WI, WID, WB, and BR. Also, AINT maintains an array of structures that store the state of the DSM system and related results, for example, the number of messages and transmission costs. The simulator works on DEC Alpha workstations running DEC/OSF1 and is upward-compatible from version 2.0.

AINT and the back-end, together, simulate a loosely coupled DSM system. Messages transmitted across the network are either control or data messages. Control messages typically contain the DSM state information, page invalidation, page upgrade, page downgrade, or page update information. Data messages contain actual page data being transferred to a requesting client. Data messages are considerably larger (1 – 4 Kbytes) than con-

Table 5. Modeled protocols and simulation parameters.

MODELED PROTOCOLS		SIMULATOR SETTINGS	
$N = 4$ SITES	$N = 8$ SITES	PARAMETER	RANGE
WI	WI	Page size (for data messages)	512 bytes; 1, 2, or 4 Kbytes
WID	WID	Control size (for control messages)	96 bytes
WB	WB	Maximum network transfer unit	1 Kbyte
BR(2 and 3, 4)	BR(2 and 3, 8)	Network size N	4 sites, 8 sites

control messages (96 bytes) and often larger than the network transfer unit of 1 Kbyte. The system maintains coherence in the simulator by transmitting control messages or data messages to sites that maintain or require copies of the page, respectively. Table 5 lists the protocols we modeled in our simulation study and the parameters used in the simulator.

As new technology emerges, we can adjust the simulation parameters to reflect the characteristics of leading-edge hardware and network environments. This, in turn, will let us project the effects of new technology on protocol behavior. As described, the simulation uses applications to generate memory reference events. An application for a DSM system is characterized by its memory access patterns. These patterns vary widely, as shown by several measurements we made (see Table 6).

The same DSM applications, written by different programmers, could potentially result in varied levels of parallelization of the problem⁹ and exhibit different communication, memory, and synchronization behaviors. This is why our DSM application suite consists of the following programs, each representing varied problems, with a broad spectrum of memory access patterns, locality, problem size, and synchronization behavior:⁸

- Parallel Matrix Multiply represents computationally intensive problems.
- Quicksort represents a class of problems that require a high level of coordination, synchronization, and management between processes, at the same time performing a large number of local operations.
- Water-NSquared is indicative of programs with more activity located at the participating clients. After the clients solve their assigned subproblems, these solutions are finally combined to solve the entire problem.¹⁰

Table 6 summarizes each program's characteristics. The data presents the

mean execution time of 20 application executions in an environment consisting of four processors. The table shows the total number of operations (reads and writes to memory), the number of read and write operations to shared memory, the sums and percentages of read and write operations to shared memory, as well as the read–write ratio of operations to shared memory. Quicksort exhibits an extremely low read–write ratio, whereas the other two applications have a very high read–write ratio; that is, reads from shared memory are much more frequent than writes to it.

Experiments and results

The simulations involved running the programs from our application suite while varying the underlying memory coherence protocol. In each simulation experiment, we ran WI, WID, WB, and the permissible instances of BR(w , n) protocols varying the DSM page size from 512 bytes to 4 Kbytes. The message model used in the simulation is more sophisticated than the one used to explain the comparison example. In particular, a 96-byte message acknowledged every message. These acknowledgments were counted as control messages. The size of a message transmitted over the network was fixed at 1 Kbyte. Therefore, depending on the size of the DSM page, up to four 1-Kbyte messages can be sent. All results presented are averaged over three simulation runs with identical settings and parameters. Parallel Matrix Multiply exhibits reproducible execution patterns on different runs. However, the Quicksort and Water-NSquared applications have less deterministic behavior.

PERFORMANCE AND FAULT TOLERANCE

Our first set of experiments examines the performance of the different DSM coherence protocols with respect to a varied degree of fault tolerance in terms of data availability.

Figure 3 shows the total number of messages transmitted (sum of data and control messages) in a DSM application using the three coherence protocols with pages sizes of 512 bytes, 1 Kbyte, 2 Kbytes, and 4 Kbytes. We count acknowledgments as control messages; this generally doubles the number of messages regardless of which coherence protocol is used. Therefore, no particular coherence protocol is favored. These observations let us qualify the total amount of traffic generated by a distributed application using DSM. The values help us estimate overhead in terms of aggregate operations to maintain memory coherence.

Figure 4 shows the number of data messages per DSM application using the three coherence protocols with page sizes of 512 bytes, 1 Kbyte, 2 Kbytes, and 4 Kbytes. Because the number of data messages varies significantly among the coherence protocols, we used a logarithmic scale. We did not count the messages used for initialization of coherence protocols, because they are not considered data messages but control messages. Furthermore, their number is negligible.

Figures 3a and 4a show the results for Parallel Matrix Multiply.

Here, for all coherence protocols, we expect that the total number of messages decreases slightly when the DSM page size is increased. We also observe this with data messages: the number of data messages *decreases* as the DSM page size *increases*. This is because, in the case of larger page sizes, a single data request results in the sending of a larger portion of the DSM segment to the issuing process. If the process writes in a sequential fashion to DSM, as Parallel Matrix Multiply does when creating the result matrix, then this leads to a decrease of data-issuing operations, thereby reducing the number of data messages. Note that with a larger page size, up to four network messages must be transmitted to install a single DSM page. Figure 4a shows that for WI and WID, the mes-

Table 6. Measured characteristics in the DSM application suite.

APPLICATION	KEY PROPERTIES	PROBLEM SIZE	TOTAL OPERATIONS	SHARED OPERATIONS	SHARED READS	SHARED WRITES	SHARED READ-WRITE RATIO
Parallel matrix multiply	Computational, intensive	64-by-64 integer matrices	567k	540×10^3 (95%)	528×10^3 (98%)	12×10^3 (2%)	44
Quicksort	Much coordination and synchronization	1,000 integers	150×10^3	14,275 (9.5%)	8,166 (57%)	6,109 (43%)	1.3
Water- NSquared	Much local activity, few global activities	512 molecules	605×10^6	157×10^6 (26%)	154×10^6 (98%)	3m (2%)	51

sage traffic is more significantly influenced by page installations. In WB and BR(w, n), page installations are fewer than in WI or WID, because the former protocols use control messages to send updates. The vast majority of messages sent by BR(w, n) and WB therefore consist of short 96-byte control messages.

The reason why BR(w, n) coherence protocols produce higher traffic than WI, WID, and WB coherence protocols (see Figure 3a) is that Parallel Matrix Multiply's read-write ratio, as stated in Table 6, is very high. In this particular case, it is 44. Thus, read operations are frequent, leading to a decrease in the number of available copies in WI and WID, thereby reducing fault tolerance and the number of coherency-related messages. WB has superior performance compared to BR(w, n) because the processors all work separately. Only a few additional copies are requested and installed over a long period of execution time, and only at the end of the application execution does the number of copies increase and the application require updates with write operations. In BR(w, n), all w installed copies have to be updated throughout the entire execution, consequently accounting for its higher costs: BR(2, 4) and BR(3, 4) coherence protocols are required to store at least two and three copies, respectively, of *any* DSM page at *any* time of the program execution. This leads to increased message traffic in case of a high read-write ratio, but also to a highly increased data availability since a certain minimal number of copies is available at all times independent of the application's current read-write pattern.

The Quicksort application exhibits a very low read-write ratio (see Table 6).

With a read-write ratio of 1.3, write operations to DSM are almost as frequent as read operations. Additionally, Quicksort requires a high degree of synchronization and cooperation among the participating processes. (We use the terms *process* and *site* synonymously in this analysis, because for these simulations, the processes were located at different sites.) Figures 3b and 4b show the results.

Because write operations are very likely and dispersed, all sites cache copies of many if not all DSM pages even in an early state of the program execution. Thus, many copies need to be updated at many sites, resulting in a large number of control messages if the DSM coherence protocol exhibits a WB-like behavior—as is the case for BR(3, 4) and, of course, WB itself. In these cases, the number of data messages is quite low. DSM coherence protocols that behave rather WI-like (such as WI, WID, and BR(2, 4)) require fewer control messages but more data messages, because certain copies must be installed at the sites at multiple times. Quicksort is a good example, showing that BR coherence protocols can naturally bridge the gap between WI and WID coherence protocols on one side and WB on the other side in terms of message costs. Because many copies are requested and installed at the beginning of the application execution, WB does not perform as well as the other coherence protocols: the read-write ratio of the application is small, and the large, increasing number of replicas must be updated frequently. Note that although the BR(w, n) coherence protocols produce lower message costs than WB in the scenario, the former guarantee a minimal degree of fault tolerance at all times, whereas the latter does not.

Figures 3c and 4c give the results of the Water-NSquared application. In Water-NSquared, once the processors receive a requested DSM page, it usually remains in possession of the requesting processor. The processors perform computations *individually* and partially accumulate information once, at the end. The read-write ratio of Water NSquared is 51—that is, very high (see Table 6). We see no substantial difference in this application in terms of total number of messages when the page size is varied, primarily because sharing is limited. The total number of messages transmitted reaches its maximum when using BR(3, 4), whereas BR(2, 4) and WB are comparable. WI and WID produce substantially fewer message costs. The reason for the high message costs of BR(3, 4) stems from the fact that even when using WB, not as many DSM copies are cached at the participating sites as for BR(3, 4). For BR(3, 4), due to the fault tolerance requirements, at least three copies of any DSM page are cached at any time.

With respect to DSM page installations and reinstallations, Water-NSquared exhibits a comparable behavior to Parallel Matrix Multiply: the larger the DSM page size, the fewer data messages are issued. In terms of the number of data messages, BR(w, n) basically lies between WI and WID on one side and WB on the other. Although BR(2, 4) and WB produce nearly the same total message costs, Figure 4c shows that BR(2, 4) reinstalls more DSM pages than WB, because the former issues approximately 9 to 15 times the number of data messages than the latter.

As a general heuristic, we can say that according to these measurements, BR(w, n) transmits more messages than WI or

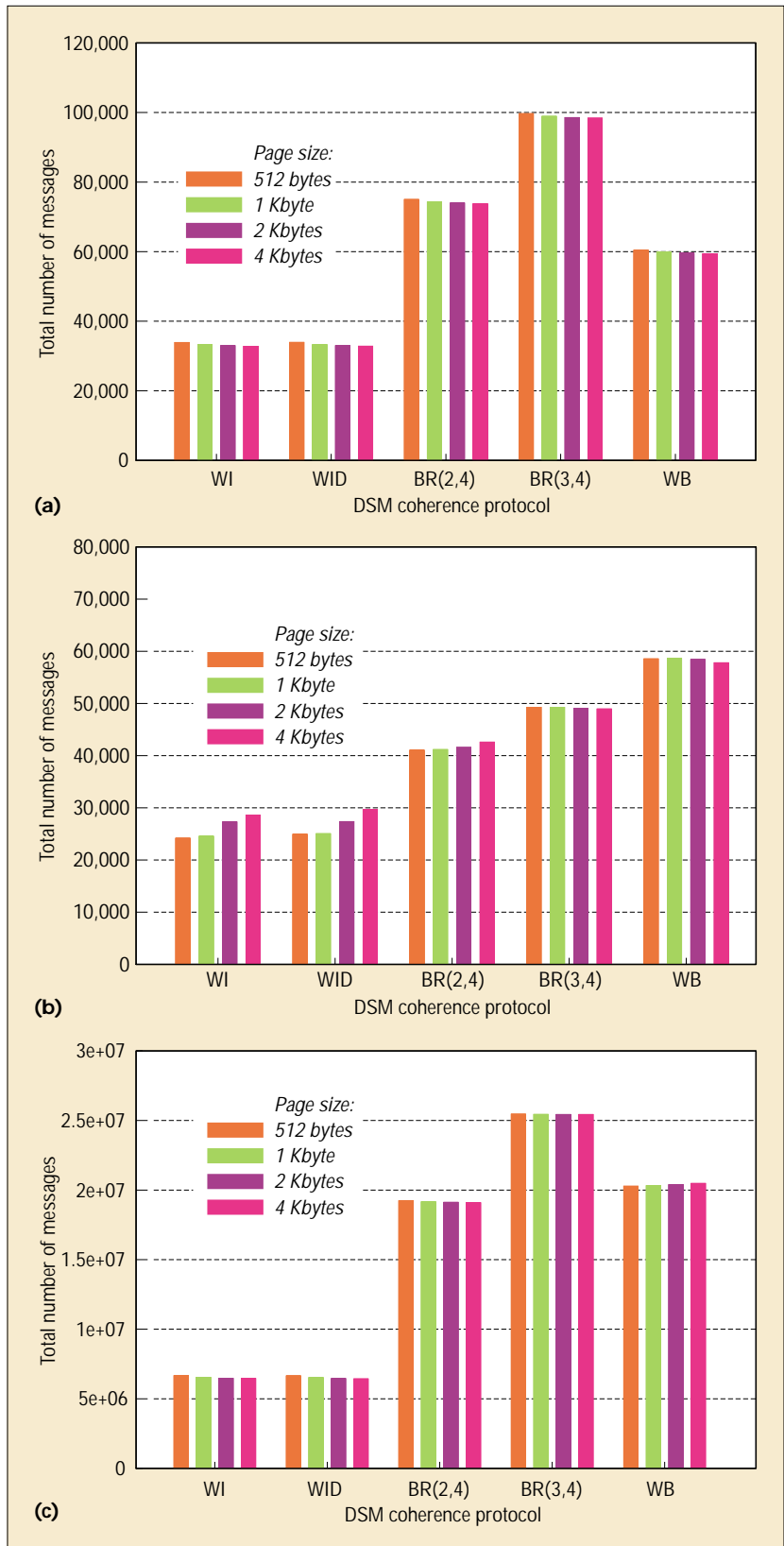


Figure 3. Total number of messages sent by WI, WID, WB, and $BR(w, n)$ coherence protocols for (a) Parallel Matrix Multiply— $BR(w, n)$ may be more expensive than even WB in terms of messages transmitted if the read–write ratio is very high; (b) Quicksort— BR typically occupies the range between WI and WB if the read–write ratio is low; and (c) Water-NSquared—varying the page size does not result in a different number of messages being transmitted, because sharing is minimal.

WID. Depending on the read–write ratio and the application’s behavior, fewer or more messages are transmitted with respect to WB. WI and WID transmit the least total number of messages. This is because of their conservative behavior—reducing the cached copies to *one* during a write operation. As a result, these protocols do not incur the high costs of keeping multiple copies of data up-to-date, but they do suffer from being highly vulnerable to failure. On the other hand, the performance in terms of message costs of the higher end of $BR(w, n)$ coherence protocols drops (approaches the performance of WB) when the read–write ratio becomes smaller, implying that the writes are occurring more frequently. Consider a $BR(7, 8)$ coherence protocol for example, and WB. While WB will continue to increase the number of cached copies as new sites request pages, it may never—depending on the particular workload—cache copies at a majority of the sites. $BR(7, 8)$, on the other hand, will force at least seven copies of each page to exist in the system throughout the application’s execution.

It is important to emphasize that neither WI, WID, nor WB guarantee that replicated copies exist *at all times*. WB potentially allows either *each* site to cache the same page, which is excessive, or just a single page copy to exist in the network, which is minimal. Clearly, any overhead incurred by $BR(w, n)$ is offset by the provision of better fault tolerance in terms of high data availability. Consequently, $BR(w, n)$ incurs competitive operation costs.

CONFIGURABILITY AND SCALABILITY

With faster networking technology and larger systems on the horizon, any mechanism or protocol that scales poorly with an increasing number of sites will result in unacceptable performance. A direct consequence of having a larger number of clients is increased communication. As a result, an important parameter to observe when investigating scalability is the total number of messages transmitted.

As already noted, WI, WID, and WB

provide no mechanism for controlling the level of data availability provided. This contrasts with the class of BR coherence protocols that allows control over the degree of fault tolerance (also called *level of replication*). Consequently, it is often possible to find an instance of BR that provides the desired degree of fault tolerance at an acceptable cost. In general, operation costs increase as the level of replication increases. The configurability offered by BR is invaluable when this class of protocols is used to provide, for example, a higher (but not excessive) level of replication—BR(4, 8) for pages integral to the system’s functioning—and a lower level of replication—BR(2, 8) for pages of lesser importance. Such configurability helps maximize the degree of fault tolerance and minimize the associated operation costs.

Configurability is provided in BR by establishing an upper bound on the number of replicas, thereby limiting the effects of scale and controlling the maximum level of replication for each page. In the second set of experiments, we varied the number of participating sites and DSM page sizes in Parallel Matrix Multiply and Quicksort and examined the behavior of WI, WID, WB, and BR(w, n). We configured the experiments to limit the upper bound on the number of copies to a level we would expect to see in real environments; our previous work³ showed that a realistic number of replicas for a page is likely to be eight or less (see Figure 1). A level of replication exceeding that increases operation costs but has a negligible impact on data availability. Thus, using this small-scale but likely configuration, our results provide some general conclusions concerning scaling the number of replicas for BR systems.

Figure 5 and Figure 6 present our findings. In each graph, the x -axis indicates the coherence protocol being examined and the variation in the number of participating sites—either four or eight (each participating site hosts exactly one participating process). For each coherence protocol, the left bar presents a network with four participating sites whereas the right bar indicates a network of eight participating sites.

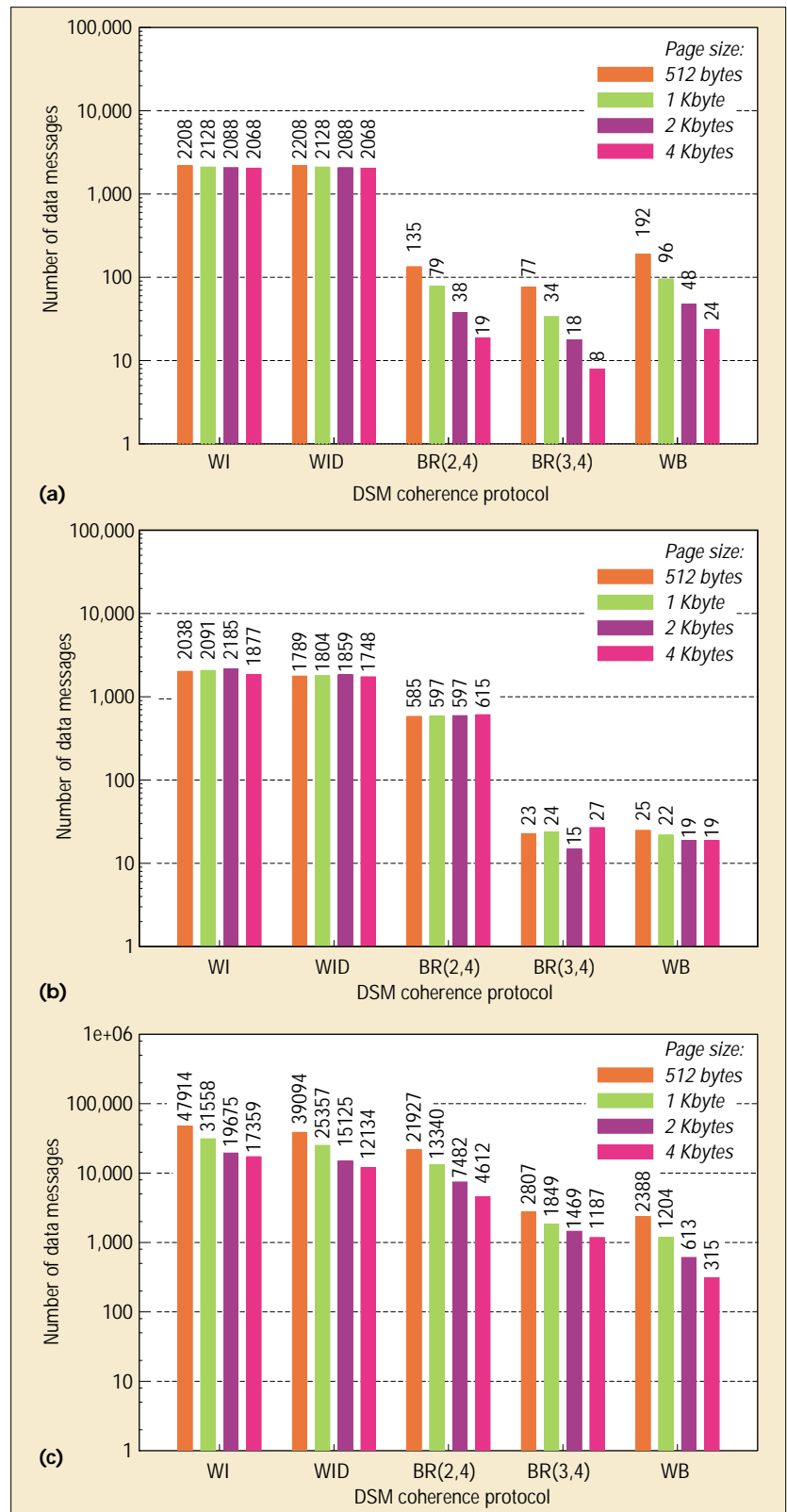


Figure 4. Number of data messages sent by WI, WID, WB, and BR(w, n) for (a) Parallel Matrix Multiply—WI and WID exhibit higher values due to repeated DSM page installations; (b) Quicksort—BR coherence protocols behave either like WI coherence protocols (for example, BR(2, 4)) with respect to DSM page reinstalls or more like the WB coherence protocols (for example, BR(3, 4)) depending on their configuration; and (c) Water-NSquared—BR coherence protocols install or reinstall more DSM pages than WB but fewer pages than WI and WID.

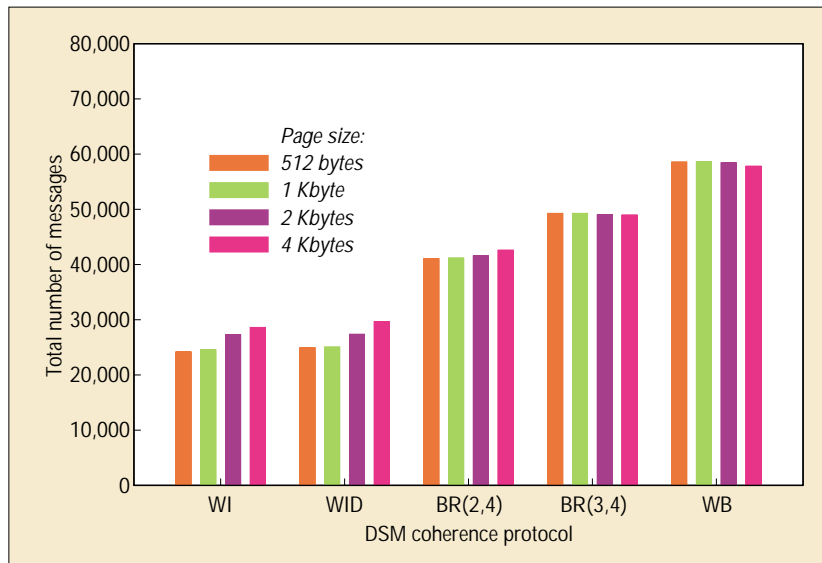


Figure 5. Scalability of WI, WID, WB, and BR coherence protocols using Parallel Matrix Multiply. The percentages indicate the increase in the total number of transmitted messages when the number of participating sites increases from four to eight.

The y -axis indicates the number of messages transmitted for each coherence protocol and network. The numbers listed above each bar of an eight-site network represent the percentage increase in the number of transmitted messages when increasing the network size from four to eight while preserving the employed DSM coherence protocol.

In examining BR, we see that the number of cached copies during a write operation can be fixed in a certain interval (by controlling W_{min} and W_{max}). As a consequence, increasing the number of sites without modifying BR's parameters w and n results in $x \in \{w, \dots, n\}$ copies of a particular DSM page to be allocated at a subset of sites $I_x \subseteq \{1, \dots, N\}$ with I_x having a cardinality of x . Depending on the application and its access pattern, x as well as I_x might change during execution. If—due to the application's access pattern—many subsequent instances of I_x differ substantially, then the cost increase is very high, because the coherence protocol must invalidate and install or reinstall a large number of DSM page copies.

If, on the other hand, w remains fixed while n is increased in correspondence with N , then the range of possible values for x increases from $\{w, \dots, n\}$ to $\{w, \dots, N\}$. Thus, the potential of subsequent instances of I_x to be substantially different is higher. This results in a further cost increase of an application if the application's access pattern is characterized by a sequence of substantially different x -values spanning a wide subrange of $\{w, \dots, N\}$ during the execution.

Figure 5 shows the results for Parallel Matrix Multiply. Although the total

number of messages of $BR(w, n)$ coherence protocols is higher than for WI, WID, and WB (due to reasons discussed earlier), the percentage increase in the total number of messages transmitted when increasing the network size from four to eight and correspondingly increasing the n parameter for $BR(w, n)$ is substantially lower than for the other coherence protocols: for $BR(w, n)$, the percentage increase is either 3% or 8%, whereas the percentage increase is 20% for WI and WID and 26% for WB. The $BR(w, n)$ coherence protocols seem to naturally support the application's access pattern: various copies are distributed among the participating sites in read phases, whereas only a few copies are maintained during write phases. Since the application can basically be characterized as a single long "read phase" (when the lines and columns of the two input matrices are read) followed by a single short "write phase" (in which the result matrix is written), local (that is, inexpensive) read operations are realized during the "read phase" and update costs are reduced by the invalidation of excessive copies at the beginning of the "write phase." This behavior results only in a slight overall cost increase—that is, in good scaling behavior.

For Quicksort, the results are summarized in Figure 6. Here, the percentage increase is 50% to 58% compared to 92%, 94%, and 105% for WI, WID, and WB, respectively. This behavior is because through its mechanism, $BR(w, N)$ has in most cases already installed copies of a DSM page when the Quicksort application needed it. Thus, DSM page

invalidations and installations are few. As a result, read operations that occur after a write operation are very likely to be local (that is, inexpensive). This leads to a reduced overall cost increase when the network (and therefore the DSM application) is scaled.

As an interesting result, for DSM applications with important access patterns such as Parallel Matrix Multiply and Quicksort, $BR(w, n)$ coherence protocols scaled significantly better than WI, WID, and especially WB for a realistic number of cooperating processes and a number of page copies likely to be used in DSM systems. In addition, $BR(w, n)$ coherence protocols were the most configurable of the protocols.

MEMORY COHERENCE PROTOCOLS greatly affect the behavior of DSM systems and govern the operations costs and number of messages transmitted. Fault tolerance and recovery methods generally depend on replicated copies of the shared data, available due to the coherence protocols.

Overall, our investigation showed that BR provides a mechanism to control the level of replication and hence the degree of fault tolerance that can be provided. The related overhead in terms of the total number of messages transmitted depends on the application's behavior. As a rule of thumb, $BR(w, n)$'s overhead is more than WI and less than WB if the DSM application exhibits a low read-write ratio—a spectrum representing a compromise between the desired degree of fault tolerance and cost. If the read-write ratio is high, $BR(w, n)$'s overhead might be even higher than the overhead of WB, especially if the application does not homogeneously reference the various DSM pages during execution. In any case and irrespective of the application's access pattern, $BR(n, w)$ is the only DSM coherence protocol that guarantees a given degree of fault tolerance in terms of data availability throughout the entire application execution.

Figure 6. Scalability of WI, WID, WB, and BR coherence protocols using Quicksort. The percentages indicate the increase in the total number of transmitted messages when the number of participating sites increases from four to eight.

WB scaled poorly, and WI and WID scaled reasonably well. BR(w, n) scaled better than WI, WID, or WB in the samples we examined while also being more configurable. Thus, if you are willing to pay for a certain degree of fault tolerance by using an instance of a BR(n, w) coherence protocol, then decreasing the DSM application's execution time by scaling the application to a higher number of processors will be rewarded by a gentle increase in terms of transmitted messages.

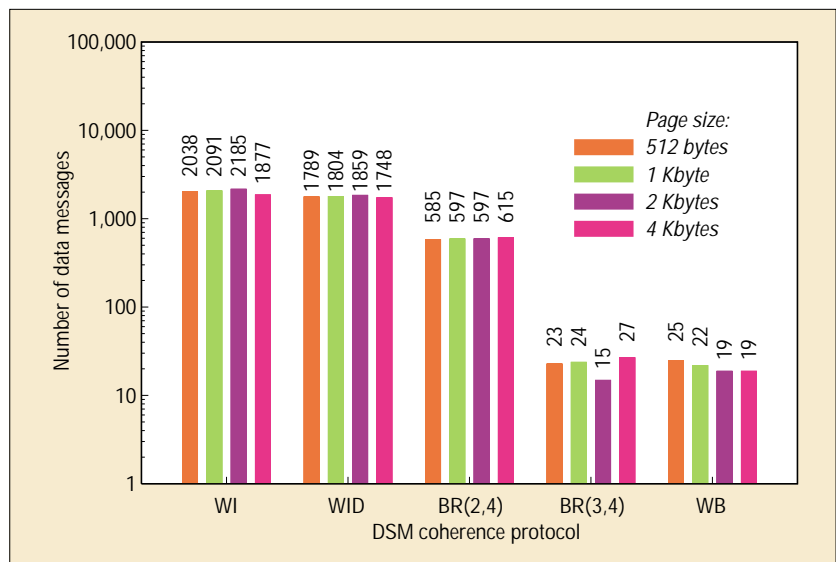
Since we conceived this article, considerable research has progressed in this area. We are implementing and measuring BR in a DSM system built under HP-UX. We are also examining policies for better selection of how and when to make copy adjustments in both the DBR coherence protocol and its extension DBRpc.¹¹ //

ACKNOWLEDGMENTS

This research was sponsored in part by US NSF grant number CCR-9704015, the UC Micro Program, and an equipment grant from HP Research Labs.

References

1. V. Lo, "Operating Systems Implementations of Distributed Shared Memory," *Advances in Computers*, Vol. 39, 1994, pp. 197-237.
2. O. Theel and B.D. Fleisch, "The Boundary Restricted Coherence Protocol for Scalable and Highly Available Distributed Shared Memory Systems," *The Computer J.*, Vol. 39, No. 6, Aug. 1996, pp. 496-510.
3. O. Theel and B.D. Fleisch, "A Dynamic Coherence Protocol for Distributed Shared Memory Enforcing High Data Availability at Low Cost," *IEEE Trans. Parallel and Distributed Systems*, Vol. 7, No. 9, Sept. 1996, pp. 915-927.
4. C. Morin and I. Pauat, "A Survey of Recoverable Distributed Shared Virtual Memory Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol. 8, No. 9, Sept. 1997, pp. 959-969.
5. M. Singhal and N. Shivaratri, "Distributed Shared Memory," Chapter 8, *Advanced Concepts in Operating Systems*, McGraw Hill, New York, 1994, pp.241-244.



6. B.D. Fleisch, N.C. Juul, and R.L. Hyde, "Mirage+: A Kernel Implementation of Distributed Shared Memory for a Network of Personal Computers," *Software Practice and Experience*, Vol. 23, No. 10, Oct 1994, pp. 569-591.
7. A. Paithankar, *AINT: A Tool for Simulation of Shared-Memory Multiprocessors*, master's thesis, Univ. of Colorado, 1995.
8. S.K. Shah and B.D. Fleisch, "A Comparison of DSM Coherence Protocols using Program Driven Simulations," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Vol. 3, CSREA Press, Athens, Ga., July 1998, pp. 1546-1553.
9. J.P. Singh, W.-D. Weber, and A. Gupta, "Splash: Stanford Parallel Applications for Shared Memory," *Computer Architecture News*, Vol. 20, No. 1, Mar. 1992, pp.5-44.
10. S.C. Woo et al., "Splash-2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Annual Int'l Symp. Computer Architecture*, ACM Press, New York, June 1995, pp. 24-37.
11. J. Turk and B. Fleisch, "DBRpc: A Highly Adaptable Protocol for Reliable DSM Systems," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems*, May 1999, pp. 340-348.

Brett D. Fleisch is associate professor of computer science and engineering at the University of California, Riverside. His research interests are in operating systems, distributed shared memory, fault tolerance, reliability, and availability; his dissertation was entitled *Distributed Shared Memory in a Loosely Coupled Environment*. He received his PhD from UCLA, his MS from Columbia University, and his BA from the University of Rochester, all in computer science. Fleisch is a member of the ACM, IEEE Computer Society, and Usenix. Contact him at Univ. of California, Riverside, Riverside, CA 92521; brett@cs.ucr.edu.

Heiko Michel is working towards his PhD at the Institute of Microelectronic Systems at the University of Kaiserslautern, Germany. His research interests include digital signal processing, especially efficient implementations of channel decoding algorithms for use in mobile communications. He received the Dipl.-Ing. degree in electrical engineering from Darmstadt University of Technology, Darmstadt, Germany. Contact him at the Univ. of Kaiserslautern, Inst. of Microelectronic Systems, Erwin-Schroedinger-Strasse, D-67663 Kaiserslautern, Germany; michel@e-technik.uni-kl.de.

Sachin K. Shah is a member of technical staff at Transarc Corporation (a division of IBM), in Pittsburgh. He received his MS in computer science from the University of California, Riverside; his thesis was entitled "Fault Tolerance and Scalability in DSM Coherence Protocols." He completed his B.Tech in computer engineering from Thadomal Shahani Engineering College, which is part of Bombay University, India. While there, Shah chaired the IEEE Computer Society chapter. Contact him at sachin@transarc.com.

Oliver E. Theel is a faculty member in the Computer Science Department at the Darmstadt University of Technology, Germany, where he earned his MSc and PhD in computer science. He spent 1994-1995 as a visiting researcher at the University of California, Riverside. His research interests are distributed systems, fault tolerance, replication techniques, properties of distributed algorithms, control theory, and self-stabilization. He is a member of the IEEE Computer Society. Contact him at Darmstadt Univ. of Technology, Alexanderstrasse 10, D-64283 Darmstadt, Germany; theel@informatik.tu-darmstadt.de.