A Dynamic Coherence Protocol for Distributed Shared Memory Enforcing High Data Availability at Low Costs*

Oliver E. Theel† Brett D. Fleisch‡

UCR-CS-95-2

*This research was partially sponsored by NSF CCR-9209405 and the DEC External Research Program.

†University of California, Department of Computer Science, Riverside, CA 92521-0304, U.S.A., and University of Darmstadt, Department of Computer Science, D-64283 Darmstadt, Germany

‡University of California, Department of Computer Science, Riverside, CA 92521-0304, U.S.A.

Abstract

Larger size networks require DSM coherence protocols which scale well. Fault-tolerance in terms of high availability is required for data access and for uninterrupted DSM service since large-scale environments have a greater number of potentially malfunctioning components. We present a new class of *dynamic* coherence protocols for DSM systems in error-prome network environments whose instances offer highly available access to DSM data at low operation costs. The approach is based on the highly scalable Boundary-Restricted (BR) coherence protocol class. The new protocol class is called the *Dynamic Boundary-Restricted* (DBR) coherence protocol class. The description of the new protocol class is accompanied by an analysis covering a large variety of workloads. This analysis presents the overall costs savings achieved by using a DBR coherence protocol instead of a static BR protocol.

Contents

1 Introduction

DSM systems have been extensively investigated during the past several years. These systems are increasingly important because of the convenient communication paradigm they provide. As a natural consequence, critical applications could potentially use DSM. For example, monitoring systems for high-risk patients or monitoring of power plants could be written based on this paradigm, leading to the need for highly reliable and highly available DSM. Although a large number of approaches have been developed (see surveys [13, 15]) only a few of these solutions have taken aspects of scalability and fault-tolerance into consideration [10, 23, 25, 7, 19]. Earlier work emphasized finding a solution to the DSM coherence problem with good performance for a relatively small network consisting of only a few sites. But as networks become larger, DSM coherence protocols which scale well and provide highly available services are necessary. *Scalability* of a DSM coherence protocol is required for a larger network with an increasing number of users (and therefore sites). *Fault-tolerance*, in terms of high availability, is required for uninterrupted DSM service even in a large-scale environments with a greater number of potentially malfunctioning components.

The instances of a new class of coherence protocols for DSM systems, called *Boundary-Restricted* coherence protocols (BR) [24], offer highly available access to shared data at low operation costs. BR protocols scale well; an increase in the number of client sites does not increase the operation costs after a certain threshold has been reached. An instance of the BR coherence protocol class is defined by two parameters: the first parameter, w, defines the minimum number of copies of a page cached at client sites. The second parameter n indicates the maximum number of client sites in the network. The BR protocol guarantees that throughout the lifespan of a distributed shared memory page, there are at least w and at most n copies available. Controlling the number of page copies in the way described above, offers two advantages: first, the degree of page data availability can be adjusted for the amount of availability required by the application. Second, the costs for accessing the page data can be varied, since the number of cached copies has a direct impact on the costs for reading or writing a replicated page.

The parameters of an instance of a BR coherence protocol are defined at the creation time of the particular DSM page, or, for a segmented DSM model, at segment creation time. In either case, the parameter values remain unchanged for the entire lifespan of the particular page. In this paper, we propose – based on the *static* BR coherence protocol class – a new *dynamic coherence protocol class*, whose instances adapt their parameters dynamically. Parameter adaptiveness offers basically an identical degree of data availability at lower operation costs compared to protocols with static parameter settings.

The remainder of the paper is organized as follows. In the next section, we give basic terminology and overview related work in the area of fault-tolerant and scalable DSM coherence protocol research. In section 3, we present our underlying model of computation. An overview of the BR coherence protocol class is presented in section 4. The subsequent section presents main analysis results covering key properties of BR coherence protocols such as operation costs, degree of availability, and behavior in a large-scale environment. These properties are exploited in section 6 where a new dynamic coherence protocol class is designed. Section 7 gives comparison results of the dynamic coherence protocol class with its non-dynamic counterpart. The last section summarizes the paper.

2 Basic Terminology and Related Work

A DSM server guarantees a particular *coherence semantics* which defines the notion of correctness and therefore *what* is guaranteed by a DSM system. Coherence semantics used in the past include release consistency [12], weak consistency [1], processor consistency [6], and sequential consistency [11]. Informally speaking, release consistency is the least restrictive coherence semantics of the list given above, allowing a high degree of parallism and superior performance characteristics by exploiting operation or application program semantics. Conversely, sequential consistency may restrict parallelism and thereby sacrifice some degree of performance. Nevertheless, its major advantage is the generality it provides: no additional semantic knowledge concerning the application program is needed by the DSM system. This is why sequential consistency is used for many DSM systems.

The coherence semantics defines the notion of correctness and therefore *what* is guaranteed by a DSM system. In contrast, the *coherence protocol* implements the coherence semantics; it provides *how* this notion of correctness is achieved. Examples for coherence protocols are write-update, write-broadcast, and write-invalidation protocols [15]. In addition, for a specific DSM system, the *coherence unit* must be defined, i.e. what abstract memory object is guaranteed to be consistent: possible objects are general application objects, segments, or single pages of a segment. For the most part, pages are often chosen, due to a variety of reasons, e.g. the identical nature, small size, and their correspondence to the memory chunks supported by the processors memory management unit. See [13, 17] for a broader discussion on this topic. In the functional model used here, we assume a DSM system guaranteeing sequential consistency for the individual pages of a segment; we define a static and a new dynamic class of fault-tolerant coherence protocols herein.

Recoverability and consistency in reliable DSM systems has been studied [10, 23, 25, 7, 19] including studies of specific systems like *Recoverable Virtual Memory* (RVM) [20, 3], *Munin* [2], *ickp* [18], and *DiSOM* [16]. Many of these studies have examined reliability from a reliable application perspective whereas our experimental work examines reliability from a memory-oriented perspective. Our experi-

mental reliability work for RELIABLE MIRAGE+ appears in [9, 8].

3 Model

In this section, we give definitions which we use throughout the paper. Additionally, we describe the functional and formal model of the DSM system under consideration.

3.1 Definitions

We assume a networking environment consisting of m sites $R_1, \ldots, R_m$ with independent failure rates. The single sites do not behave in a Byzantinian manner, i.e. they either function according to their specification or not at all. The sites are arbitrarily connected by communication links which may also fail independently from each other and independently from the sites. This may result in network partitions with several independently operating subnets. In addition, we assume that within the network, a distributed shared memory service is available which functions in the manner described in the next section. The DSM system maintains one or more *segments* $S_1, \ldots, S_s$. A segment S_i consists of one or more *pages* $P_1, \ldots, Pp_i$, and finally all of those pages consists of a certain number of basic memory units, e.g. bytes or words. Whereas the number of pages per segment might vary, the size of the basic memory units per page is the same for all segments. Let $\mathbf{C} \subset \{R_1, \ldots, R_m\}$ be the set consisting of all potential client sites of the DSM system, with $\mid \mathbf{C} \mid = n < m$ giving the cardinality of this set. $S \in \{R_1, \ldots, R_m\}$ is the DSM server site. Only sites included in $\mathbf{C}$ are allowed to submit requests to the DSM server S.

3.2 Functional Model of a DSM System

Aside from operations for creating and destroying segments, the most important operations that a (segmented) DSM system must support are *read* and *write operations* to a particular page of a segment. These operations are carried out using the migratory paradigm [22]: the client site which wants to read (write) a specific page currently not available in the local memory submits a *read request (write request)* to the *DSM server* which maintains the segment control information. The DSM server is a designated site of the network owning data structures and implementing protocols which enable it to carry out the requested operations. In particular, the DSM server stores the request queue and keeps directory information for the pages in the network. For the sake of generality, a DSM server can potentially also be a client, eliminating network communication for requests (e.g. $S \in \mathbf{C}$). In the ordinary case, clients and server are different sites, thus the requests must be transmitted from the clients to the server using the communication infrastructure. Once a client has submitted its request (i.e. it received an acknowledgment from the DSM server), it blocks and waits for the *result* delivered from the server. Analogously, results are either submitted to the client with or without the use of the communication

links, depending on whether the client and the DSM server reside on the same site or on different ones. The result of a request typically consists of the requested data and a *mode*. In both cases, the requested operation is regarded as being *successful*, because the client received desired results. Due to the state of the network or the state of the DSM system itself, it might not always be possible to carry out the requested operation. For example, the site which stores the page may not be reachable. In this case, an error message is returned and the request is deemed *unsuccessful*. It is, among others, the aim of this work to construct DSM systems where the vast majority of requests are carried out successfully. This leads to highly available DSM.

Once a client received the results of a successful request, which includes the data stored on the page, it maps the page data into its memory and uses it only according to the granted mode. The page is deemed *cached* in a particular mode with respect to a particular operation. The mode determines how subsequent read and write operations originating at this client site operate. Figure 1 gives an overview of the various modes used by the functional model. A *mode* is regarded as being a two-dimensional

Mode	**Description**
(local read, local write)	the page data can be read and modified locally, i.e. without the need to submit read or write requests to the DSM server
(local read, global write)	the page data can be read locally, but write operations must be submitted to the DSM server in order to execute a global write operation
(global read, global write)	neither a read nor a write operation can be executed locally. This mode is implicitly assumed for all pages which are not cached at a client site
(global read, local write)	not used

Figure 1: *Modes associated with DSM pages at the client sites*

tuple consisting of a *read attribute* in the first dimension and a *write attribute* in the second dimension. The mode instances are stored at the client sites. The read (write) attribute defines how a read (write) operation originating at a particular client site must be handled: If the client wants to read (write) the data of a page cached in a mode containing a local read (local write) attribute, then this results in a *local read operation* (*local write operation*) carried out locally on the client site. If a particular page is not cached at all or cached in a mode including a global read (global write) attribute, then a read (write) request must be submitted to the DSM server, triggering a global read operation (global write operation) performed by the server, as described earlier in this section.

4 Boundary-Restricted Coherence Protocol Class

We briefly present the basic idea, the underlying concept, and characteristic properties of the BR coherence protocol class in this section. Our discussion is based on the functional model described previously.

4.1 Fault-Tolerance Background

The key concept for achieving fault-tolerance in terms of high availability is *redundancy*. The higher the degree of redundancy the more malfunctioning system components can be tolerated. Applying this concept to DSM systems requires increasing the number of page copies (also called *replicas*) cached at sites in the network. Clearly, there is a price to be paid: replicas require management overhead which includes keeping multiple copies up-to-date or preventing the client from reading out-dated copies.

Informally speaking, we call the number of sites contacted by the DSM server during the execution of a read (write) operation *read (write) operation costs*. The probability that at an arbitrary point in time the data of a DSM page can be accessed is called *(data) availability*. The management approach has a strong impact not only on the costs of subsequent operations but also on the data availability. For instance, when updating *all* existing replicas, the number of up-to-date (or *actual*) copies of a page remains constant. Thus, the availability of the DSM page is not affected. The same holds true for the costs: since the number of copies and their storage locations has not changed, subsequent read and write operations may be carried out at the same costs. This is not the case when a write operation on a page alters the number of replicas: deleting replicas has a severe impact on the availability and the costs of subsequent operations. If, for example, a write operation decreased the number of copies from 5 to 1, and the single site caching page subsequently fails, then the page data becomes unavailable. On the other hand, reducing the number of replicas from 5 to 1 reduces the costs of the next write operation, since only one copy must be updated instead of five. This discussion shows that there is a strong correlation between operation costs and data availability.

4.2 Design Policy

As suggested earlier, operation costs affect the operation availabilities and vice versa. It is important to understand factors that have an impact on those parameters. Clearly, the coherence protocol used by the DSM system strongly influences the operation costs.

As a summary of observations from the discussion above, a suitable coherence protocol for a large, error-prone distributed environment must have the following properties:

(P1) Limited Workload Dependability - the number of copies of a page should, only to a certain degree, be dependent on the workload, i.e. the mixture of read and write operations

(P2) Lower Bound in the Number of Cached Copies - situations where only a single copy of a page exists should be avoided. The availability of the page data is extremely vulnerable to single component failures in this case.

(P3) Upper Bound in the Number of Cached Copies - situations where all the client sites cache a copy of a page and *all* pages must be updated as part of a global write operation should

be avoided. The probability that a write operation can be executed successfully decreases if the number of caching sites increases significantly. This is because more potentially error-prone components are involved and sucessful completion of the write is less likely.

The conceptional mechanism used by the BR coherence protocol which guarantees these design policies is based on the following observation:

A DSM server's workload can be divided into two different types of phases. The first phase starts with the execution of the first global read operation after a global write operation and ends with the next global write operation. The second phase starts with the first global write operation following a global read operation and continues until the execution of the next global read operation.[1] We call the former phase *read phase* since it consists entirely of read operations and the latter phase analogously *write phase*. Figure 2 depicts alternating read and write phases as they are encountered by a DSM server processing requests for a particular DSM page. In order to control the number of page copies present in

Figure 2: *Alternating read and write phases as monitored by the DSM server*

the critical situations (P1) – (P3) stated above, we establish the following constraints. We refer to these constraints as *protocol invariants* since the coherence protocol must ensure them at all points in time (analogous to the sequential consistency coherence semantics, which must also always be guaranteed).

Protocol Invariants

The BR coherence protocol guarantees that during a read phase, copies of a page are exclusively cached in (local read, global write)-mode, and that the number of cached copies N_r during this phase always satisfies the constraint:

$$r_{min} \leq N_r \leq r_{max} \qquad \text{with} \qquad r_{min}, N_r, r_{max} \in \mathbf{N} \tag{1}$$

with r_{min} (r_{max}) representing the minimum (maximum) number of cached copies in the read phase. $\mathbf{N}$ is the set of positive integers excluding zero. For the number of copies present and modified as part of the execution of a local or global write operation N_w in the write phase, the following always holds:

$$w_{min} \leq N_w \leq w_{max} \qquad \text{with} \qquad w_{min}, N_w, w_{max} \in \mathbf{N} \tag{2}$$

[1]Note that the local read and local write operations cannot be monitored by a DSM server since they are not resulting in a request submitted to the server.

With these invariants, Figure 3 shows three cases which correspond to coherence protocols well-known from the literature. Part a) of the figure shows the boundary setting for the Write-Broadcast coherence

Figure 3: *Boundary Settings for three coherence protocols: a) Write-Broadcast, b) Write-Invalidate, and c) Write-Invalidate with Downgrading*

protocol (WB) [15], part b) for the Write-Invalidate coherence protocol (WI) [15], and part c) shows the boundary setting for the *Write-Invalidation with Downgrading* coherence protocol (WIwD) [5, 4]. The WIwD coherence protocol is similar to the WI protocol, except when a global read occurs and the requesting site gets an actual copy, then the copy of the former writer's page is *downgraded* from mode (local read, local write) to (local read, global write). Since the page is not deleted, it increases the *minimum* number of cached copies in the read phase to two. Unfortunately, even this approach suffers from too few copies present in the write phase. The BR coherence protocols address this serious drawback by increasing the minimum and maximum number of copies modified by a write operation (i.e. required in the write phase) together with the minimum number of copies cached in the read phase by $w - 1$ with $w \in \{2, 3, \ldots\}$. This leads to boundary settings given in Figure 4. These constraints are

Figure 4: *Boundary Settings for the BR coherence protocol*

enforced by the DSM server while serving read or write requests, and as part of the initial configuration when a segment is created. The values for r_{min}, r_{max}, w_{min}, and w_{max} can ideally be different for every page, leading to a different degree of availability for the pages' data and to different operation costs. As already indicated, a coherence protocol with boundary settings as specified in Figure 4 is called *Boundary-Restricted* (BR) coherence protocol, since it restricts the number of cached copies to lie between w_{min} and r_{max}. Furthermore, because $w_{min} = w_{max}$ and $r_{min} = w_{max} + 1$, it suffices to instantiate w_{min} and r_{max}. They are also referred to as w and n, the number of potential client sites, respectively. An instance of the BR coherence protocol with given values for w and n is addressed as

BR(w, n) coherence protocol.

4.3 Protocol Mechanism

In this section, we describe the particular protocol operations. They represent the mechanism which enforces the underlying policy.

Segment creation operation

Segment creation is performed at the DSM server after the client specifies the segment size, initial values for all the pages (eventually in an implicit manner, like initialize_with_zeroes), and values for the two boundaries w and n for all the pages. The segment creation operation can be regarded as a special write operation to all the pages of the new segment. Since page copies do not exist yet, the DSM server must create at least w replicas with the initial values on at least w client sites. Usually, one set of all replicas is installed on the site where the segment creation request originates. If the installation of the required number of copies succeeds, then the operation is successful and the DSM server informs the client. If this is not possible, or if w is greater then $\mid \mathbf{C} \mid$ (i.e. there are less client sites than required for enforcing the lower boundaries of this particular coherence protocol for at least a single page), then the segment creation operation fails. In this case, the DSM server immediately notifies the client about the unsuccessful attempt. A pseudo-code specification of the segment operation is given in Figure 5.

```
create_DSM_segment (in: client_site, sgmt_nbr, PL, PV) {
    — create a new DSM segment sgmt_nbr by installing a minimal number of page copies
    — with initial values. Examples for PL and PV are:
    — PL =< (2,3), (3,4) > and PV =< (0...0), (0...0) >
    — (number of pages of the segment is length of page list PL)
    nbr_pages = | PL |
    C = < set of potential client sites >
    page_mode = (local_read, global_write)
    for page_nbr = 1 to nbr_pages do {
        (w, n) = cut_list_head (inout: PL)
        if (w > | C |)
            — lower boundary cannot be guaranteed
            signal (unsuccessful)
        — select client sites which should cache a copy of the page
        — (strategy can be much more sophisticated)
        CC ∈ {A ∪ {client_site} | A ⊆ (C \ {client_site}) ∧ | A | = w − 1}
        — get initial values of the page
        page_data = cut_list_head (inout: PV)
        add_page_entry_to_directory (in: sgmt_nbr, page_nbr, (w, n))
        for all new_client_site ∈ CC do {
            install_page_copy (in: new_client_site, sgmt_nbr, page_nbr, page_data, page_mode )
            add_owner_entry_to_directory (in: new_client_site, sgmt_nbr, page_nbr, page_mode)
        }
    }
}
```

Figure 5: *Procedure for installing a new segment*

Read operation

If a process at a client site wants to read a particular data item from a page, then the memory management unit checks whether this page is cached and therefore mapped into the local memory. Furthermore, the system verifies whether the local read attribute is part of the mode associated with the page. If both are true, then the read operation can be performed locally. Otherwise, a read request must be send in a message from the client site to the DSM server for the associated segment. Once the read request is received at the DSM server and ready to be executed, then the steps in Figure 6 are executed.

```
serve_DSM_read_request (in: sgmt_nbr, page_nbr, client_site) {
    — serve DSM read request once it has become the head element of the request
    — queue for the particular segment and page. (The request was enqueued
    — earlier and spent some time waiting to be served). Note that the DSM
    — server is in read phase when processing the current read request.
    CC = directory_lookup (in: sgmt_nbr, page_nbr, "set-of-copy-owners")
    if (client_site ∈ CC)
        — client site already owns a current copy; disregard request
        return
    nbr_copies = | CC |
    (w, n) = directory_lookup (in: sgmt_nbr, page_nbr, "boundary-values")
    if (nbr_copies + 1 > n) {
        — destroy one copy before proceeding
        victim_site ∈ CC
        destroy_page_copy (in: victim_site, sgmt_nbr, page_nbr)
        delete_owner_entry_in_directory (in: victim_site, sgmt_nbr, page_nbr)
    }
    .
    .
    .
    < serve read request as usual >
    .
    .
    .
}
```

Figure 6: *Procedure for serving a read request*

Write operation

The client site first checks whether the write operation performed locally. In order to do so, a copy of the required page must have been cached and mapped into the site's memory. Furthermore, a local write attribute belonging to this page must be present. If one or both of those requirements does not hold, then a message containing a write request must be sent from the client site to the DSM server. Once received, the DSM server performs the steps given by the algorithm of Figure 7. At the end of the write operation it is guaranteed that only current page data is cached at the sites storing replicas.

4.4 An Example of a BR Coherence Protocol

Figure 8 demonstrates how the BR(2,5) coherence protocol functions under a sample workload for one page. The set of client sites is given by $\mathbf{C} = \{R_1, \ldots, R_5\}$, thus all the sites can potentially cache a copy of the page. Since $\mid \mathbf{C} \mid = n = 5$, the upper boundary n on number of client sites cannot be exceeded

```
serve_DSM_write_request (in: sgmt_nbr, page_nbr, new_page_data, client_site) {
    — serve DSM read request once it has become the head element of the request
    — queue for the particular segment and page. (The request was enqueued
    — earlier and spent some time waiting to be served). Note that the DSM
    — server is in write phase when processing the current write request.
    C =< set of potential client sites >
    CC = directory_lookup (in: sgmt_nbr, page_nbr, "set-of-copy-owners")
    nbr_copies = | CC |
    VC = { }
    (w, n) = directory_lookup (in: sgmt_nbr, page_nbr, "boundary-values")
    if (client_site ∈ CC) {
        if (nbr_copies > w) {
            — destroy v copies before proceeding
            v = nbr_copies − w
            VC ∈ {A | A ⊆ (C \ {client_site}) ∧ | A | = v}
        }
    }
    else { — client_site ∉ CC
        if (nbr_copies + 1 > w) {
            — destroy v copies before proceeding
            v = nbr_copies − w + 1
            VC ∈ {A | A ⊆ C ∧ | A | = v}
        }
    }
    for all (victim_site ∈ VC) do {
        destroy_page_copy (in: victim_site, sgmt_nbr, page_nbr)
        delete_owner_entry_in_directory (in: victim_site, sgmt_nbr, page_nbr)
    }
    ⋮
    < serve write request as usual >
    ⋮
    if (client_site ∉ CC) {
        page_mode = (local_read, global_write)
        install_page_copy (in: sgmt_nbr, page_nbr, page_data, page_mode, client_site)
        add_owner_entry_to_directory (in: client_site, sgmt_nbr, page_nbr, page_mode)
    }
}
```

Figure 7: *Procedure for serving a write request*

throughout this particular scenario.

Figure 8: *Number and location of copies for the BR(2,5) coherence protocol*

The example gebins with the execution of a read request issued by R_2. After the read, R_2 together with R_1 and R_4 cache a copy of the page. (R_1 and R_4 were arbitrarily chosen). In a further step, site R_5 attempts to read the page. Since it does not cache a copy with a local read attribute, the execution of the corresponding global read operation by the DSM server increases the number of copies in the network. After R_2 locally reads its copy of the page (the local read attribute is present), site R_4 submits a write request to the DSM server. Because $N_r = 4 > w = 2$ as defined by the coherence protocol, the DSM server must – prior to the start of the write operation – destroy at least two cached copies. In this example, it chooses the copies located on sites R_2 and R_5.[2] Only R_1 and R_4 own a copy containing the most recent data, once the write operation has successfully been completed. A subsequent global write operation executed on behalf of R_3 leads to the destruction of the copy held by R_4. This was necessary, since the maximum number of copies present in a write phase would have exceeded w while executing the write operation. Finally, a subsequent read operation issued by site R_1 can be performed locally.

The fundamental difference between the WB and the WI as well as the WIwD coherence protocols can be observed when analyzing the states encountered before and after the execution of the write operation issued by R_4. In case of using the WB coherence protocol, all existing copies would have been updated after the write operation, whereas by using the WI or WIwD coherence protocols, all copies except the copy on site R_4 would have been invalidated and destroyed. The BR(2,5) coherence protocol on

[2]The decision which copies to delete can be based on various policies: analogous to load balancing algorithms these decisions can be extremely simple or very sophisticated, e.g. by taking static or even dynamic network information into account. Since the particular policy has no impact on the correctness of the coherence protocol presented here, we do not consider this issue further.

the other hand, mitigates the high costs for this write operation occurring when compared to the WB coherence protocol and mitigates the tremendous reduction in data availability when compared to the WI or WIwD coherence protocols. It does so by increasing costs against a gain of availability and decreasing availability against a saving in costs. The example shows that the BR coherence protocol is a hybrid combination of the WB and WIwD coherence protocols. On one hand, for $w = 1$ the BR coherence protocol is identical to the WIwD approach. On the other hand, the protocol uses the WB approach for those sites which either plan to modify the page or already cache a read copy. The next section presents analysis results of the BR coherence protocol class in terms of operation costs and data availability.

5 Analysis of the BR Coherence Protocol Class

The properties of DSM coherence protocols which are of primary interest are the read and write *costs*, and the *availability* of the user data managed by the DSM system (and therefore managed by the coherence protocol). They are formally defined as follows:

Definition 5.1 (Costs) Let CP be the coherence protocol used by the DSM server in a DSM system. The number of sites contacted by the DSM server using the communication infrastructure of the network in order to carry out a global read (write) operation are called *read (write) costs.* □

This definition of costs does not include the costs which arise from submitting the request to the DSM server. This additional cost would lead to an increase of one copy when the client and the server are not co-located. The definition was chosen to represents the part of the overall cost which is expected to vary when using different coherence protocols. The costs for submitting requests to the DSM server occur regardless of the nature of the coherence protocol chosen.

Definition 5.2 (Data Availability) Let CP be the coherence protocol used by the DSM server of a DSM system. The probability A_{CP}^{data} that at an arbitrary point in time at least one functioning site caches an actual copy of a page is called *data availability (of this particular page by using* CP*).* □

The analysis of the BR coherence protocol class is based on a stochastic model using the Markov chain given in Figure 9. The ovals present the states of the markov chain. The states ending with a "R" stand for those states one can encounter in a read phase, while the one ending with a "W" represents the state of the write phase. The preceding figure (e.g. "$w + 1$") indicates the number of copies cached on client sites. Thus, for example, the state labeled with "$w + 1$R" represents the system state where $w + 1$ copies are distributed to $w + 1$ client sites while the DSM server is monitoring a read phase. "wW" represents the only state of the write phase. In this case, exactly w copies are cached at w different client sites. The arrows between the ovals represent possible state transitions. The weight of an arrow indicates the probability that the system switches from the state the arrow originates at

Figure 9: *Markov chain used for deriving the steady state distribution of the BR(w, n) coherence protocol*

(tail of the arrow) to the state the arrow ends at (head of the arrow). For example, the probability that the system goes from state "wW" to state "$w+1$R" is given by p_{gr}. We refer to those weights as *transition probabilities*. The transition properties used in the Markov chain analysis are given in Figure 10. The Markov chain analysis allows to determine the *steady state distribution* of the DSM system

Probability	**Description**
p_{lr}	Probability that a site already caching a page re-reads the data. It depends on the read attribute as to whether this can be done locally (local read attribute) or whether the site must issue a read request (global read attribute). In the BR coherence protocol, as previously described, read operations originating from a client site owning a copy are always satisfied locally.
p_{gr}	Probability that a site currently not caching a copy of a page requests the data.
p_{lw}	Probability that a site already storing a copy of a page wants to modify it. In this case, if a local write attribute is present, this can be done locally. If the global write attribute is set, then the site must submit a write request to the DSM server. In the BR coherence protocol described here, write operations are always executed globally, since no sites have a local write attribute.
p_{gw}	Probability that a site currently not caching a copy of a page wants to modify the page's data.

Figure 10: *Transition probabilities used for the Markov chain analysis*

using a BR(w, n) coherence protocol. This distribution describes the probability that an observer finds the DSM system in a particular state. The distribution is the basis for calculating important measures like the mean read and write costs $\overline{E}^{read}_{\mathsf{BR}}$ and $\overline{E}^{write}_{\mathsf{BR}}$, the mean operation costs $\overline{E}^{op}_{\mathsf{BR}}$, and the mean number of cached copies $\overline{E}^{nbr}_{\mathsf{BR}}$. The results are given below:

$$\overline{E}^{nbr}_{\mathsf{BR}}(w) = \mathcal{Q} + w \tag{3}$$

$$\overline{E}^{read}_{\mathsf{BR}}(w) = \frac{p_{gr}}{p_{lr}+p_{gr}} \cdot \left[\frac{w-1}{\mathcal{N}\cdot\mathcal{P}} + 2\right] \tag{4}$$

$$\overline{E}^{write}_{\mathsf{BR}}(w) = \frac{p_{lw}}{p_{lw}+p_{gw}} \cdot [\mathcal{Q}+w] + \frac{p_{gw}}{p_{lw}+p_{gw}} \cdot [\mathcal{Q}+(w+1)] \tag{5}$$

$$\overline{E}^{op}_{\mathsf{BR}}(w) = (p_{lr}+p_{gr}) \cdot \overline{E}^{read}_{\mathsf{BR}}(w) + (p_{lw}+p_{gw}) \cdot \overline{E}^{write}_{\mathsf{BR}}(w) \tag{6}$$

with

$$\mathcal{P} = \frac{p_{gr}}{1 - p_{lr}}, \tag{7}$$

$$\mathcal{N} = \frac{1}{\mathcal{P}} + \mathcal{P}^{n-w-2} \cdot \frac{p_{gr}}{1 - p_{lr} - p_{gr}} + \frac{1 - \mathcal{P}^{n-w-1}}{1 - \mathcal{P}} \tag{8}$$

and

$$\mathcal{Q} = \frac{1}{\mathcal{N}} \cdot \left[\frac{1 - \mathcal{P}^{n-w-1}}{1 - \mathcal{P}} + \mathcal{P} \cdot \frac{1 + (n-w-2) \cdot \mathcal{P}^{n-w-1} - (n-w-1) \cdot \mathcal{P}^{n-w-2}}{(1 - \mathcal{P})^2} \right]. \tag{9}$$

A remarkable fact is, that the behavior of all those cost measures is almost linear when w is increased. Except in those cases, where the difference between the lower and the upper boundary, i.e. $n - w$, is relatively small, the mean number of cached copies and the mean write costs behave non-linearly. (The same holds for the mean operation costs, since it is directly impacted by the mean write costs.) The mean read costs behave nearly linearly. They are *de facto* not affected by the size of $n - w$. The reason for this behavior can quite easily be identified: the terms $\mathcal{N}$ and $\mathcal{Q}$ contain several geometric sums of the form $\sum_{i=0}^{n-w-2} \mathcal{P}^i$ and $\sum_{i=0}^{n-w-2} i \cdot \mathcal{P}^i$. Since $\mathcal{P} < 1$, these sums have a known limit and reach it very rapidly when $n - w$ grows:

$$\lim_{n \to \infty} \mathcal{N} = \frac{1}{\mathcal{P}} + \frac{1}{1 - \mathcal{P}} \tag{10}$$

$$\lim_{n \to \infty} \mathcal{Q} = 1 / \lim_{n \to \infty} \mathcal{N} \cdot \left[\frac{1}{\mathcal{P}} + \mathcal{P} \cdot \frac{\mathcal{P}}{(1 - \mathcal{P})^2} \right] = \frac{p_{gr}}{p_{lw} + p_{gw}}. \tag{11}$$

As a consequence, any further increase of $n-w$ does not have an impact on the steady state distribution; therefore, it has no impact (or only a very limited one) on the mean costs and mean number of copies. These properties have significant consequences with respect to availability and scalability issues.

As already mentioned, the mean costs and the mean number of cached copies behave in a linear manner, assuming the difference between n and w is sufficiently large. In the following, we present linear equations for those cost functions. These approximations are sufficiently precise, once $n - w$ is larger than 10. The approximations simplify the analysis of the BR protocol class since linear equations, in contrast to equations of higher-order polynomials, can be investigated more easily. In particular, they ease the identification of major dependencies among the values involved, thereby simplifying for instance the design of a BR coherence protocol with partially given characteristics. The approximations are given below.

$$\lim_{n \to \infty} \overline{E}_{\mathsf{BR}}^{nbr}(w) = 1 \cdot w + \frac{p_{gr}}{p_{lw} + p_{gw}} \tag{12}$$

$$\lim_{n \to \infty} \overline{E}_{\mathsf{BR}}^{read}(w) = \frac{p_{gr}}{p_{lr} + p_{gr}} \cdot \left[\underbrace{\frac{1 - p_{lr} - p_{gr}}{1 - p_{lr}}}_{a_{gr}} \cdot w + \underbrace{\frac{1 - p_{lr} + p_{gr}}{1 - p_{lr}}}_{b_{gr}} \right] \tag{13}$$

$$\lim_{n \to \infty} \overline{E}_{\mathsf{BR}}^{write}(w) = \underbrace{1}_{a_{write}} \cdot w + \underbrace{\frac{p_{gr} + p_{gw}}{p_{lw} + p_{gw}}}_{b_{write}} \tag{14}$$

$$\lim_{n\to\infty} \overline{E}^{op}_{\mathsf{BR}}(w) \quad = \quad \Big(p_{gr}\cdot a_{gr} + (p_{lw}+p_{gw})\cdot a_{write}\Big)\cdot w + \Big(p_{gr}\cdot b_{gr} + (p_{lw}+p_{gw})\cdot b_{write}\Big) \qquad (15)$$

5.1 Availability

In this section, we investigate the degree of *data availability* provided by the BR(w,n) coherence protocol class. The mean number of cached copies is used to determine the probability the DSM system is able to deliver the most recent value of a particular page. As already pointed out, the greater the average number of replicas present in the network, the higher the probability that data can be returned by the DSM server. In Figure 11, selected mean numbers of cached copies are given assuming that there are 25 client sites in the network. For comparison, we also give the corresponding values for the WB and WI coherence protocol. The local/global operation ratio is 1/9. Obviously, the WB protocol caches the

Mean Number of Cached Copies											
n	rwr	p_{lr}	p_{gr}	p_{lw}	p_{gw}	BR$(1,n)$	BR$(2,n)$	BR$(3,n)$	BR$(4,n)$	WI	WB
25	8	0.05	0.84	0.05	0.06	8.1	9.1	10.1	11.0	7.3	25
25	4	0.05	0.75	0.05	0.15	4.7	5.7	6.7	7.7	3.9	25
25	2	0.05	0.62	0.05	0.28	2.8	3.8	4.8	5.8	2.2	25
25	1	0.05	0.45	0.05	0.45	1.9	2.9	3.9	4.9	1.4	25
25	0.5	0.05	0.28	0.05	0.62	1.4	2.4	3.4	4.4	1.1	25
25	0.25	0.05	0.15	0.05	0.75	1.2	2.2	3.2	4.2	1.0	25
25	0.125	0.05	0.06	0.05	0.84	1.1	2.1	3.1	4.1	1.0	25

Figure 11: *Mean number of cached copies in the WI, the WB, and the BR coherence protocol*

maximum number of copies in all cases. This is generally true, but unfortunately, the WB coherence protocol achieves this highly available behavior at a significant cost: the mean operation costs for any reasonable value of p_{lw} are extremely high. The price for achieving this degree of data availability is not only too high but also not necessary, as we will show later in this section. As expected, the mean number of cached copies of the BR(w,n) protocol is essentially equal to the corresponding value of the BR$(w+1,n)$ protocol plus one for $w = 1,2,3$. The figures for the BR coherence protocols remain basically unchanged even when the number of client sites n increases. This is due to the limit property discussed in the previous section.

In order to determine how many replicas are actually needed to achieve a certain degree of data availability in a particular network with failure-bound components, we present Figure 12. We assume that all sites of the network – or at least those caching copies – are functioning with a uniform probability of p. This implies that the individual sites are up and running and also that the software needed to manage the copy is in working order. Since a page data is available in the network as long as at least one of $c > 0$ copies is accessible, the data availability is computed according to a binomial distribution

Figure 12: *Data availability depending on the number of cached copies and the probability that sites are functioning*

as

$$A_{data}(p) = 1 - (1-p)^c. \tag{16}$$

Figure 12 shows the data availability graphs for six different values of p. From the graph of $A_{data}(0.75)$ for example, it can be determined that with only one replica present in the system, the data availability is, of course, 0.75 (lower left corner of the figure). By increasing the number of replicas to two, a data availability gain of 18% is achieved. If three replicas are guaranteed, then the data availability increases to approximately 98%, and once more than five copies are cached, the data availability is nearly 100%. The major result of the analysis is, that for realistic values of $p > 0.75$ [14], four replicas are sufficient to guarantee data availability. Thus, those coherence protocols which offer a mean number of cached copies of at least four, at the lowest cost possible, are of particular interest. The approximation of the BR coherence protocol given in section 5 permits the reader to identify those protocols easily.

6 A Dynamic Coherence Protocol Class

We now present an extension to the "static" BR coherence protocol class to which we refer to as the *Dynamic Boundary-Restricted* (DBR) coherence protocol class. The motivation for constructing such a class of dynamic protocols stems from the following observations:

(O1) Long-term Workload - The workload of a DSM server varies tremendously from application to application.

(O2) Short-term Workload - The workload submitted from a currently running application to the DSM server is highly varied throughout the run-time.

Unfortunately, it is difficult to identify a single, optimally suited coherence protocol used by the DSM server for a variety of different applications since there is a strong impact from the application-generated workload on DSM system performance. Building a DSM server which performs reasonably well for arbitrary long-term[3] workloads is complex. Solving the short-term workload problem is no less complex.

It is possible to conceive of protocols that can be adapted to solve this problem. However, even when it is possible to use different coherence protocols on an application-to-application basis, the decision of which particular protocol to use must be made based upon known workload characteristics *prior* to the execution of the application. Switching to a different coherence protocol *while* the application is being executed presents significant challenges since the protocols involved might have to maintain different management information and use different assumptions with respect to the number and placement of page copies. Furthermore, since in most applications the workload in terms of read and write requests submitted from the application to the DSM server strongly varies over time, an *a priori* chosen coherence protocol can only be optimal *on the average* for a given, assumed short-term workload. There is no guarantee that – although on the average correct – due to dynamic behavior, some constraints like the minimal degree of data availability is violated during certain time periods.

The dynamic coherence protocol class presented in this section mitigates these difficulties. We address the problem associated with the long-term workload by using a general coherence protocol. This general coherence protocol class (or *framework*) represents a *parameterized coherence protocol.* The advantage of using such an unifying approach in contrast to a hybrid approach represented by e.g. a collection of different coherence protocols are several. First, the framework needs only be implemented once in contrast to multiple implementations of various coherence protocols. Second, a single implementation is time-saving, more robust, and can be easily maintained. Third, if the framework allows us to model different coherence protocols, then a well-suited instance can be chosen on an application-to-application basis. Fourth, since only an adjustment of the framework parameters is required in order to switch from one well-tuned system to another one, this adjustment can easily be done even while the application is running. This mechanism can be exploited to address the short-term workload problem. Fifth, because the control structures required by the framework remain the same – regardless of the particular parameter setting – minimal reconfiguration work must be performed for switching between instances. As a consequence, the DSM server using instances of such a dynamic coherence protocol framework can easily and frequently adapt to changing application characteristics.

[3]We use the terms "long-term" and "short-term workload" since the problems associated with these remind us of similar problems known as "long-term" and "short-term scheduling" in the area of operating systems' process management.

6.1 Objectives of Dynamic Behavior

As mentioned earlier in this paper, the main goal is to develop a coherence protocol which offers high data availability. As demonstrated, high availability is achieved by increasing the number of cached copies of a page throughout the run-time of the application. Unfortunately, a greater number of cached copies also means higher operation costs. Since for most real-world networks it can be assumed that the probability that a site and the software running on it functions with a probability of at least 75%, there is no need to guarantee more than four replicas. In such a case, the availability of the page data is almost perfect, i.e. close to 100%. Any further increase in the number of replicas would merely increase the operation costs. Once a desired degree of data availability can be technically achieved by a coherence protocol, the question remains, whether it is possible to "tune" the protocol in such a way that is provides the desired data availability at the lowest operation costs possible. To summarize the discussion, the dynamic behavior of a coherence protocol CP should guarantee the following properties:

$$A_{\mathsf{CP}}^{data} \geq a \qquad \text{(Main goal)} \tag{17}$$

$$\overline{E}_{\mathsf{CP}}^{op} \rightarrow \text{Min} \qquad \text{(Sub goal)} \tag{18}$$

with a being the required lower bound of the data availability.

By using the BR coherence protocol class as a starting point for constructing such a dynamic coherence protocol class, these constraints can be re-written by using equations (3), (6), and (16):

$$1-(1-p)^{\left\lfloor \overline{E}_{\mathsf{BR}}^{nbr} \right\rfloor} \geq a \tag{19}$$

$$\overline{E}_{\mathsf{BR}}^{op} \rightarrow \text{Min} \tag{20}$$

By substituting the precise formulas for calculation the mean operation costs and the mean number of cached copies by their limits if n goes to infinity, these constraints can be substantially simplified. Again, we would like to emphasize that this simplification does not result in imprecise results once the difference between the n and w parameters of the BR protocol class is approximately 10 or larger. Thus, using equations (12) – (16) the constraints are

$$1-(1-p)^{\left\lfloor \lim_{n\to\infty} \overline{E}_{\mathsf{BR}}^{nbr}(w) \right\rfloor} = 1-(1-p)^{\left\lfloor \frac{p_{gr}}{p_{lw}+p_{gw}}+w \right\rfloor} \geq a \qquad \text{and} \tag{21}$$

$$\begin{aligned} &\lim_{n\to\infty} \overline{E}_{\mathsf{BR}}^{op}(w) \\ &= p_{gr} \cdot \left[\left(1 - \frac{p_{gr}}{1-p_{lr}} \right) \cdot (w-1) + 2 \right] + p_{lw} \cdot \left[\frac{p_{gr}}{p_{lw}+p_{gw}} + w \right] + p_{gw} \cdot \left[\frac{p_{gr}}{p_{lw}+p_{gw}} + (w+1) \right] \\ &\rightarrow \text{Min.} \end{aligned} \tag{22}$$

6.2 Controlling Parameters

The parameters involved in equations (21) and (22) can be divided into two classes: one class of parameters is given by the networking environment, the application, and availability policy. These

parameters are pre-set and cannot be altered by the coherence protocol. The second class of parameters consists of those parameters which can be freely chosen among certain implicitly given boundaries. The parameters of the former class are $p_{lr} + p_{gr}$, $p_{lw} + p_{gw}$, a, and p whereas the only parameter of the latter class is w – a parameter of the BR coherence protocol.[4] Since w is present in all of those equations, altering w leads to a variation of the mean number of cached copies and the mean operation costs. Thus, whenever a change in the parameters of the networking environment, the availability policy, or the short-term application workload is encountered, it is possible for the protocol to react. This is performed by altering the current the value of w in such a way that the constraints given by equations (21) and (22) are still guaranteed (see Figure 13).

Figure 13: *Parameters which influence the data availability and the operation costs. By adjusting w, given constraints regarding the data availability and the operation costs can be guaranteed even when other parameters vary over time.*

6.3 Solution to the Optimization Problem

After identifying w as the only controlling parameter of $\lim_{n\to\infty} \overline{E}^{op}_{\mathsf{BR}}(w)$ and A^{data}_{BR}, we now present a solution to the optimization problem.

In order to satisfy equation (21), the following must hold true:

$$w \geq \max\left\{\left\lceil \log_{1-p}(1-a) - \left\lfloor \frac{p_{gr}}{p_{lw}+p_{gw}} \right\rfloor \right\rceil, 1\right\} \qquad \text{with } p, a < 1 \tag{23}$$

Any value of w satisfying this equation leads to a data availability of at least a. For a solution of equation (22) we must identify the value of w among those values of w satisfying equation (21) which leads to minimal mean operation costs. This is very simple, since the mean operation costs behave in a monotonic increasing manner while w is growing. Thus, the minimal value of w satisfying equation (21) also solves equation (22). It is therefore a solution to the entire optimization problem. Figure 14 gives values for w required to maintain a data availability of 90%, 95%, 98%, and 99%. The table shows that

[4]The second parameter of the BR protocols, namely n, cannot be used to alter the protocol properties once $n - w$ is beyond a certain threshold. An increase of n, for example, has no effects on the mean operation costs or the mean number

Minimal w for Maintaining given Degree of Data Availability																
					$p = 0.75$; w if A_{BR}^{data} is				$p = 0.85$; w if A_{BR}^{data} is				$p = 0.95$; w if A_{BR}^{data} is			
rwr	p_{lr}	p_{gr}	p_{lw}	p_{gw}	90%	95%	98%	99%	90%	95%	98%	99%	90%	95%	98%	99%
8	0.05	0.84	0.05	0.06	1	1	1	1	1	1	1	1	1	1	1	1
4	0.05	0.75	0.05	0.15	1	1	1	1	1	1	1	1	1	1	1	1
2	0.05	0.62	0.05	0.28	1	2	2	3	1	1	2	2	1	1	1	1
1	0.05	0.45	0.05	0.45	2	3	3	4	2	2	3	3	1	2	2	2
0.5	0.05	0.28	0.05	0.62	2	3	3	4	2	2	3	3	1	2	2	2
0.25	0.05	0.15	0.05	0.75	2	3	3	4	2	2	3	3	1	2	2	2
0.125	0.05	0.06	0.05	0.84	2	3	3	4	2	2	3	3	1	2	2	2

Figure 14: *Minimal values of w for guaranteeing data availability of 90%, 95%, 98%, or 99% based on equations (16) and (23)*

the lower the read/write ratio rwr and the lower the availability of the components of the networking environment (sites and software) p, the higher is the minimal value of w. In other words: the greater the number of write operations in a network consisting of highly error-prone components, the more copies of a page must be provided in order to provide a certain degree of data availability. On the other hand, the more read operations performed in the workload of a very reliable, available application, the less copies of a page are needed for a given degree of data availability. This corresponds with our intuitive understanding of the data availability problem.

6.4 From BR(w,n) to DBR(w)

In this section, we describe the extensions necessary to the BR(w,n) coherence protocol class in order to construct the DBR(w) coherence protocol class. (The parameter w of the DBR protocols – analogous to the w-parameter of the BR(w,n) protocols – it indicates the guaranteed minimum number of page copies of a single page present in the network.) In particular, we specify:

1) how a decision about an adjustment is made, what kind of adjustment is made, and how is it enforced,
2) what kind of dynamic data required for initiating an adjustment is collected,
3) where this data is collected, where the decision about an adjustment is made, and finally how often decisions and adjustments are made.

6.4.1 Decision Making and Adjustment Enforcement

Issue 1) is straightforward: for every page of the distributed shared memory segment, there exists a value for the parameter w. Once a more recent value of w according to equation (23), referred to as

of cached copies.

w^*, has been calculated, w and w^* are compared. In case they differ, the rule given in Figure 15 is used. Whenever the number of replicas in a write phase (corresponding to the w parameter) is smaller

```
initiate_adjustment_of_w (in: w*, sgmt_nbr, page_nbr) {
    w = directory_lookup (in: sgmt_nbr, page_nbr, "w-value")
    if (w = w*)
        — nothing must be done
        return
    if (w > w*)
        — the number of cached copies is unnecessarily high
        destroy_multiple_cached_copies (in: sgmt_nbr, page_nbr, w − w*)
    else
        — the number of cached copies must be increased
        create_multiple_cached_copies (in: smgt_nbr, page_nbr, w* − w)
    directory_update (in: sgmt_nbr, page_nbr, "w-value", w*)
}
```

Figure 15: *Decision making of the DBR(w) coherence protocol class*

then required, $w^* - w$ replicas must be artificially created on arbitrary client sites currently not caching copies. On the contrary, whenever w is larger than actually required, then $w - w^*$ replicas *can* be destroyed without violating the availability constraint. Since we are also interested in having as small operation costs as possible, such a reduction of replicas should be provided as quickly as possible. The corresponding actions to be taken are given in Figures 16 and 17.

```
create_multiple_cached_copies (in: sgmt_nbr, page_nbr, nbr_new_copies) {
    C = < set of potential client sites >
    CC = directory_lookup (in: sgmt_nbr, page_nbr, "set-of-copy-owners")
    PC = C \ CC
    NC = select_set_of_new_cache_sites (in: PC, nbr_new_copies)
    owner_site = select_owner_site (in: CC)
    page_data = get_page_data (in: owner_site)
    page_mode = (local_read, global_write)
    for all (client_site ∈ NC) do {
        install_page_copy (in:, client_site, sgmt_nbr, page_nbr, page_data, page_mode)
        add_owner_entry_to_directory (in: client_site, sgmt_nbr, page_nbr, page_mode)
    }
}
```

Figure 16: *Procedure for creating additional page copies*

6.4.2 Dynamic Data Collection

Issue 2): in order to allow the calculation of a new, optimal value for w, the values of a, p as well as of p_{gr} and $p_{lw} + p_{gw}$ must be known. a, the degree of data availability, is given by the system or application designer. In the former case, a is fixed for quite a long period of time, if not forever. In the latter case, a is constant for at least the run-time of the particular application, i.e. a can be regarded as constant

```
destroy_multiple_cached_copies (in: sgmt_nbr, page_nbr, nbr_victims) {
    CC = directory_lookup (in: sgmt_nbr, page_nbr, "set-of-copy-owners")
    if (| CC | − nbr_victims < 1)
        — do not delete all copies
        nbr_victims = | CC | − 1
    VC = select_victim_set_of_cache_sites (in: CC, nbr_victims)
    for all (victim_site ∈ VC) do {
        destroy_page_copy (in: victim_site, sgmt_nbr, page_nbr)
        delete_owner_entry_in_directory (in: victim_site, sgmt_nbr, page_nbr)
    }
}
```

Figure 17: *Procedure for destroying unnecessary page copies*

with respect to the short-term workload. p quantifies the availability of a single site of the network.[5] We assume that this value is identical for all the sites. This restriction is not essential for our solution of a dynamic coherence protocol, but simplifies the presentation. A generalization for varying values of p for the single site of a network is trivial. Thus, the only data which must be *dynamically* collected are current values of p_{gr} and $p_{lw} + p_{gw}$.

Unfortunately, the value of p_{gr} not only depends on the particular application, but also on the coherence protocol which manages the DSM. For example, a read operation issued under control of one coherence protocol A might result in a global read operation: a read request, since local page data is not available for read access. On the other hand, under control of a coherence protocol B, local data (and a local read attribute) may be available. Thus, in the former case, the read operation must be executed globally, whereas in the latter case it can be performed as a local operation. Consequently, the probability p_{lr} is different while executing the same application but with different protocols. Even when repeatedly executing the same application assuming an identical initial configuration, this workload parameter is likely to be subject to changes, due to the fact that sites may fail, resulting in varying placements of cached copies. This is in contrast to the sums $p_{lr} + p_{gr}$ and $p_{lw} + p_{gw}$ which are *not* affected by the properties of the coherence protocol used. They are fully determined by the application. As a consequence, the precise values of p_{gr}, p_{lw}, and p_{gw} can only be derived *after* the execution of the application. We will show later on in this paper how we can circumvent those obstacles with our approach.

The following equations show how p_{gr} and the sum $p_{lw} + p_{gw}$ are principally computed:

$$p_{gr}^{\Delta} \quad := \quad \frac{n_{gr}^{\Delta}}{n_{lr}^{\Delta} + n_{gr}^{\Delta} + n_{lw}^{\Delta} + n_{gw}^{\Delta}} \tag{24}$$

[5] It is also possible to take advantage of a statistical process which collects data with respect to the availability of the network components. This process could be contacted periodically in order to get more recent values for p. In this paper, however, we focus on workload changes and not changes due to e.g. the aging of network components.

$$p_{lw}^{\Delta} + p_{gw}^{\Delta} \quad := \quad \frac{n_{lr}^{\Delta} + n_{gw}^{\Delta}}{n_{lr}^{\Delta} + n_{gr}^{\Delta} + n_{lw}^{\Delta} + n_{gw}^{\Delta}} \tag{25}$$

n_{gr}^{Δ} (n_{gw}^{Δ}) is the number of global read (write) requests issued and n_{lr}^{Δ} (n_{lw}^{Δ}) the number of local read (write) operations which occurred within a certain time window Δ – in this case during the run-time of the application. (If Δ is set to the entire run-time period of an application, we omit the Δ-index.) The major drawback of a static coherence protocol which is optimized with respect to those probabilities is, that – although on the average optimal – there are time periods within Δ where the coherence protocol is either far to expensive or does not guarantee the desired degree of data availability. To illustrate this, we give the following sample scenario:

Consider an application using DSM with the BR(w, n) coherence protocol. Assume availability of the network components is $p = 0.75$. Assume further that $p_{lr} = p_{lw} = 0.05$, $p_{gr} = 0.52$, and $p_{gw} = 0.38$. The dynamic behavior of a sample application is given in Figure 18. The application can be partitioned

Section	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Average	
$p_{lr}^{\Delta_i}$	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	p_{lr}	0.05
$p_{gr}^{\Delta_i}$	0.63	0.60	0.57	0.53	0.48	0.41	0.32	0.41	0.48	0.53	0.57	0.60	0.63	p_{gr}	0.52
$p_{lw}^{\Delta_i}$	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	p_{lw}	0.05
$p_{gw}^{\Delta_i}$	0.27	0.30	0.33	0.37	0.43	0.49	0.58	0.49	0.43	0.37	0.33	0.30	0.27	p_{gw}	0.38
$grwr^{\Delta_i}$	2.00	1.75	1.50	1.25	1.00	0.75	0.50	0.75	1.00	1.25	1.50	1.75	2.00	$grwr$	1.21
rwr^{Δ_i}	2.16	1.89	1.63	1.37	1.11	0.84	0.58	0.84	1.11	1.37	1.63	1.89	2.16		

Figure 18: *Dynamic behavior of an application*

into 13 sections of equal length. For every section, approximations for p_{lr}, p_{gr}, p_{lw}, and p_{gw} calculated based on the corresponding number of requests encountered in the current section Δ_i are given. $p_{gr}^{\Delta_i}$, as an example, was calculated according to the following formula:

$$p_{gr}^{\Delta_i} = \frac{n_{gr}^{\Delta_i}}{n_{lr}^{\Delta_i} + n_{gr}^{\Delta_i} + n_{lw}^{\Delta_i} + n_{gw}^{\Delta_i}} \qquad i = 1, \ldots, 13. \tag{26}$$

If we assume a minimal data availability of 93.75% (as guaranteed by having at least two copies of a page cached), w used by the BR protocol must be set to at least two, since $grwr := p_{gr}/(p_{lw} + p_{gw})$ is calculated as 1.21. This also determines the mean operation costs of 2.37 according to equation (15). When calculating the data availability and the mean operation costs using the corresponding values of each section and building their weighted sum with respect to the section length, we obtain a data availability of 90.14% and mean operation costs of 2.37. Those values reflect more precisely the situation encountered during the run-time of the application. Thus, it turns out that – although the mean operation costs are equal in this scenario – the required lower bound in terms of data availability is violated. Specifically, on the average, there are *less* than two copies of a page available, leading to an unsatisfactory data availability. In fact, in this example, the standard deviation of the number of

cached copies encountered during the run-time of the application is 9.19%. It can be calculated that for 23.1% (Δ_6, Δ_7, and Δ_8) of the run-time, the data availability is only 75%, thus substantially less than required. In particular, on the average while availability is seemingly acceptable there are periods of time where availability falls below the minimum permitted. This is due to the *dynamic* behavior of the workload and the *static* setting of w for achieving a desired degree of data availability.

By using the dynamic variation approach of w throughout the run-time of an application this serious drawback can be overcome. The dynamic DBR(2) coherence protocol applied to the scenario given above guarantees a data availability of 94.47% on the average during the run-time. The required lower bound 93.75% of the data availability is never violated. The mean operation costs are slightly greater than those of the static protocols, namely 2.55 in contrast to 2.37. This is an increase of 7.6% in terms of costs using 2.37 as base or a decrease of 7% when using 2.55 as base: obviously the price to be paid in order to get a) a data availability increase of 4.33% on the average and b) a 100% guarantee of the lower data availability constraint of 93.75%.

The example, although giving an idea of the advantages achieved using a dynamic coherence protocol, is imprecise in that it divides the run-time of the application Δ into several disjunct partitions Δ_i. For every one of those partitions, it is assumed that the probability $p_{gr}^{\Delta_i}$ and the probability of a write request $p_{lw}^{\Delta_i} + p_{gw}^{\Delta_i}$ are known at the beginning of the particular interval. This is generally not the case, since the required values for the number of requests $n_{lr}^{\Delta_i}$, $n_{gr}^{\Delta_i}$, $n_{lw}^{\Delta_i}$, and $n_{gw}^{\Delta_i}$, needed for deriving those values according to equation (26), can only be determined *after* the partition has ended. Even calculating the ratios while a particular partition is forthcoming does not help: in this case, the ratios calculated at the beginning of a new partition would be highly imprecise due to the fact that they must be computed based on only a few requests. As a consequence they are likely to not be characteristic for the current partition. The closer to the end of the partition, the more precise the ratios become. At the end of the interval, they are in correspondence with the true values. Such behavior is certainly not desirable because the adaptation mechanism may trigger too many boundary changes during the unstable starting phase of every new partition. A possible solution to this problem is *prediction* of the upcoming workload based on the workload observed in the past. The calculation of *exponential averages* [21] is such an approach used in the area of short-term CPU scheduling.

A closely related approach, though, is to not having defined multiple partitions in the first place, for which the workload parameters must be calculated, but instead *moving averages* for calculating them "on the fly". In this case, the entire run-time of the application forms a single, large partition. The moving averages for those parameters can either be built using a certain real-time window, for example *the last t time units*, or using a virtual-time window, i.e. *the last r requests submitted to the DSM server.* Using the former method, the parameters are calculated as follows. (We only give the formulas for the

probability derived at time t of read requests originating at a client site currently not caching a page copy, namely $p_{gr}^{t,\Delta}$. The other cases are calculated accordingly.):

$$p_{gr}^{t,\Delta} := \frac{n_{gr}^{t,\Delta}}{n_{lr}^{t,\Delta} + n_{gr}^{t,\Delta} + n_{lw}^{t,\Delta} + n_{gw}^{t,\Delta}} \tag{27}$$

Δ is the real-time window used. $n_{gr}^{t,\Delta}$ ($n_{gw}^{t,\Delta}$) is the number of read (write) requests issued by clients not owning a copy during the last Δ real-time units prior to the calculation of the value. Analogous cases hold for $n_{lr}^{t,\Delta}$ and $n_{lw}^{t,\Delta}$. Here the client sites hold page copies. The calculation of the parameters using the virtual-time window approach is based on the following equation.

$$p_{gr}^{t,N} := \frac{n_{gr}^{t,N}}{N} \tag{28}$$

N is the virtual-time window (of N requests received at the DSM server) and $n_{gr}^{t,N}$ the number of read requests issued by clients without copies within the time window N at time t. In the following section we restrict the discussion to the virtual-time window based approach

6.4.3 Active Entity enforcing Adjustment

Issue 3) questions are strongly related. The DSM server is the only entity in the network which is able to monitor the number and nature of distributed shared memory request issued without cost. This site maintains a pending request queue and directory information about what client is currently caching which page of a segment. Thus, the DSM server site is the only site that knows about all global read and write requests.

The maintenance of up-to-date approximations for *grwr* using either the real-time or virtual-time window approach, involves very low cost, since the data necessary to compute them is accessible at the DSM server site anyway: whenever a new request for a specific page is received by the DSM server, it automatically updates the corresponding counters (for example incrementing $n_{gr}^{t,N}$ for an incoming global read request and decrementing the appropriate counter of the operation received "N requests earlier"). Subsequently, a new approximation for *grwr* must be calculated. Since incoming DSM requests are typed as either being reads or writes, the DSM server can directly decide whether or not to increment the counter for n_{gr}.[6] In case the submitted request is a write, then there is no need to distinguish between write requests originating from clients sites with or without cached copies (although possible by comparing it with the directory maintained at the DSM server), since for calculating *grwr* only the sum of p_{lw} and p_{gw} is required.

Unfortunately, due to the inability of the DSM server to observe the number of local read operation $n_{lr}^{t,N}$, the *probability* $p_{gr}^{t,N}$ and the *probability sum* $p_{lw}^{t,N} + p_{gw}^{t,N}$ cannot be calculated. This does not deem the

[6] In the DBR protocol class, no read requests submitted to the DSM server have been originated from a client site currently caching a copy, thus all read requests are global.

entire approach to be unsuccessful, because the overall goal is to derive the current value of $grwr$ and not the probabilities themselves. An approximation for $grwr$, abbreviated as $grwr^{t,N}$, can be derived without observing a current value for $n_{lr}^{t,N}$; only $n_{gr}^{t,N}$ and $n_{w}^{t,N} := n_{lw}^{t,N} + n_{gw}^{t,N}$ must be kept up-to-date:

$$grwr^{t,N} := \frac{p_{gr}^{t,N}}{p_{lw}^{t,N} + p_{gw}^{t,N}} = \frac{n_{gr}^{t,N}}{n_{lw}^{t,N} + n_{gw}^{t,N}} = \frac{n_{gr}^{t,N}}{n_{w}^{t,N}} \tag{29}$$

A pseudo-code implementation of this procedure is given in Figure 19. After a new value of w has

```
derive_new_value_of_w (in: client_nbr, request_type, sgmt_nbr, page_nbr out: w*) {
    if (request_type = "read") {
        — read request issued by a client site holding a page copy
        n_gr^{t,N} = n_gr^{t,N} + 1
        enqueue (inout: time_window_queue, in: "global-read")
    {
    else { — request_type = "write"
        n_w^{t,N} = n_w^{t,N} + 1
        enqueue (inout: time_window_queue in: "write")
    }
    if (length (in: time_window_queue) = N + 1) {
        — dequeue request "N virtual time units ago" if queue is full
        old_request_type = dequeue (inout: time_window_queue)
        if (old_request_type = "global-read")
            n_gr^{t,N} = n_gr^{t,N} − 1
        else — old_request_type = "write"
            n_w^{t,N} = n_w^{t,N} − 1
    }
    — calculate new value for w
    grwr^{t,N} = n_gr^{t,N} / n_w^{t,N}
    w* = max { ⌈log_{1−p}(1 − a) − ⌊grwr^{t,N}⌋⌉ , 1 }
}
```

Figure 19: *Procedure for gathering statistics and calculating a new value of w*

been computed, the DSM server immediately makes a decision according to the rule given in Figure 15. Thus, the required adjustment of the number of cached copies is enforced immediately.

7 Analysis of the DBR(w) Coherence Protocol Class

In this section, we compare instances of the DBR(w) coherence protocol class with the static counterparts of the BR(w, n) coherence protocol class. The analysis is intended to demonstrate 1) that the dynamic coherence protocols meet their design goals and 2) under which workload they perform well.

Figure 20 gives – beyond others – the behavior of the approximation of $grwr$ calculated during the execution of a DSM application (part a of the figure). The DBR(3) coherence protocol is used in this example. Thus, it is possible to adopt a protocol behavior either according to the BR($1, \infty$), the BR($3, \infty$), or the BR($3, \infty$) coherence protocol, depending on the nature of the workload encountered during the execution of the application. Assuming a probability of $p = 75\%$ that the sites of the network are functioning correctly, this guarantees minimal data availability of 93.75% (see Figure 12 for other

Figure 20: *Adaptation of the mean number of cached copies throughout the run-time of an application*

values of p and their associated data availabilities). Since according to equation (12), the mean number of cached copies is the sum of the value of w and the actual value of $grwr$, the summation of the two values given by a) and b) for every point in time during the application execution gives the mean number of cached copies at this particular point in time. For example, there are approximately 3.4 replicas until the time represented by the center of section c of part c) of the figure present in the network. By setting the w boundary statically to three – as a static BR coherence protocol would be restricted to – there are several intervals in which the required minimal mean number of cached copies of three is exceeded, resulting in unnecessarily high operation costs. Figure 20 depicts excessively high costs in sections b, d–f, and h–l. The DBR coherence protocol lowers, in exactly those cases, the value w in such a way, that the summation of $grwr$ and w is at least equal to the minimal number of cached copies. Since there must be at least one copy of a page present in a write phase (and this not only on the average but guaranteed), w can never drop below one. Thus, even if the value for $grwr$ is equal or even greater than the required minimum number of cached copies, as it is the case in section j, the DBR *cannot* adjust the currently enforced value of w to zero. In those cases, the mean number of cached copies is extremely high, resulting in very high operation costs.

Part c) of Figure 20 shows the resulting behavior of the mean number of cached copies provided by the DBR(3) coherence protocol. A change in the current value of w is required precisely at those points in time where the DSM server observes a value of $grwr$ of either 1, 2, or 3. Depending on whether $grwr$ is increasing or decreasing until that time, the DSM server lowers or raises w. Because an adaptation only takes place at those points in time, we refer to them as *adaptation points.*[7] Because the current

[7]This definition of adaptation points is independent of the desired level of data availability: the points fencing section j in the given example are – according to the definition – also adaptation points, although no adaptation takes place if the

value of $grwr$ changes only at adaptation points and because only "full" copies of a page contribute to the data availability[8] (if e.g. $\lim_{n\to\infty} \overline{E}^{nbr}_{\mathsf{DBR}}(w) = 2.3$ then there are only two "full" copies cached on the average), the precise behavior of $grwr$ between adaptation points has no impact on the data availability encountered in such a section. This, however, does not hold true for the mean operation costs where the lower the graph of $grwr$ is within a given section, the lower the mean operation costs.

In order to simplify the subsequent discussion, we restrict the behavior of $grwr$ over time t, indicated as $grwr(t)$, to "staircase functions". It is obvious that every "step" (or section) of such a staircase with height $grwr(t)$, adaptation points S and E, $S < E$, and $grwr(t) = \lfloor \min\{S, E\} \rfloor$ for all $t \in [S, T]$ is representative for all possible temporal behaviors of $grwr$ over sections of the same length, i.e. length $E - S$, with $grwr(t)$ varying arbitrarily between $\min\{S, E\}$ and $\max\{S, E\}$ with respect to the degree of data availability provided throughout such a time window. The mean operation costs derived by restricting the analysis to staircase functions are the lowest possible ones for the degree of data availability.

When comparing instances of the DBR with instances of the static BR coherence protocol class, we have not considered the costs which arise in the DBR for actually performing the adaptation. The costs for such an adaptation in terms of number of remote sites contacted are typically either one or two, since $grwr(t)$ can only increase or decrease by one between adaptation points. Thus, either one copy must be deleted or a new copy must be created. In the former case, a site caching a copy must be informed to delete it. In the latter case, two client sites might be involved in the operation due to the fact that 1) an actual copy must be contacted and 2) its value forwarded to the new caching site. Since the number of adaptation points encountered during the run-time of an application is negligible compared to the number of read and write requests, this simplification does not have a significant impact on the outcome of the analysis.

These assumptions have two major consequences: first, there are only finite different values of $grwr(t)$ and second, the particular sequence in which those values are adopted over time has no impact on the results of the analysis. Consequently, two applications having the same $grwr(t)$ values over time periods of the same length, but not necessarily at the same time, can be treated equally in the analysis. Therefore, since the timing information is not crucial, we can omit it. An appropriate representation of an application's workload behavior can therefore be given as a histogram stating what values of $grwr(t)$ were adopted for what percentage of the application's run-time. Figure 21 gives the workload representation of the sample application of Figure 20. Graph a) is the original workload behavior in

minimal number of cached copies is set to three. If this number is set to four, though, then adaptation will be enforced even at those points.

[8]Note, that the dense function for data availability is discrete.

Figure 21: *Reduction of the workload behavior to staircase functions. Since timing information is obsolete, workload behavior can be presented as a histogram*

terms of $grwr$. The staircase function incurring the minimal operation costs is given by graph b). Since not exact timing information but only the lengths of the several sections are of interest (see part c), the entire workload behavior can be presented as a histogram (part d). This histogram shows e.g. that in 31.7% of the run-time of the application (assuming a staircase-like workload behavior) $grwr(t)$ equals one. Note that the histogram can also be regarded as discrete dense function stating with a probability of 31.7% an observer monitoring the workload of an application encounters $grwr(t)$ being one.

The assumptions given above do not favor any of the coherence protocols investigated as part of the following analysis, but they tremendously reduce the number of different workload behaviors to examine.

Figure 22 gives the results of analytical comparisons between instances of the BR coherence protocols with their dynamic counterparts on the basis of eight sample workloads. (The workloads are given in form of discrete dense functions as discussed above.) We chose the DBR(2) and DBR(3) protocols and compared them with the BR$(2, \infty)$ and BR$(3, \infty)$ protocol, respectively. By setting the number of potential client sites of the BR protocols to $n = \infty$, we indicate that the approximations for the costs measures are used instead of the precise values. This allows a fair comparison with the corresponding instances of the DBR protocol class. For every protocol, the mean read and mean write costs, the mean number of cached copies, and mean operation costs are given. Furthermore, the figure shows the corresponding data availability for site and software availabilities of 75%, 90%, 95%, 98%, and 99%. The different workloads chosen have in common that the ratio $grwr$ varies at most between zero and eight. If $grwr(t) = 0$, then until that particular time during the run-time of a sample application, there are no global read operations contained in the workload. If $grwr(t) = 1$, the workload consists of as many write requests as global read requests. As $grwr(t)$ increases, more global read requests form part of the workload. If $grwr(t) = 8$, then, on the average, there are eight global read requests for every single write request.

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	1.8495	6.7368	6.0000	2.9361	99.07%	99.88%	99.97%	$\approx$100%	$\approx$100%
DBR(2)	1.6610	5.8480	5.1111	2.6069	98.38%	99.77%	99.94%	$\approx$100%	$\approx$100%
BR$(3, \infty)$	2.0380	7.7368	7.0000	3.3708	99.77%	99.99%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.7112	6.0702	5.3333	2.7917	99.42%	99.97%	$\approx$100%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	1.7182	4.8421	4.0000	2.9557	98.33%	99.78%	99.95%	$\approx$100%	$\approx$100%
DBR(2)	1.4817	4.0421	3.2000	2.5562	97.09%	99.58%	99.90%	$\approx$100%	$\approx$100%
BR$(3, \infty)$	1.9547	5.8421	5.0000	3.5453	99.58%	99.98%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.5722	4.4421	3.6000	2.8887	98.96%	99.94%	99.99%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	2.0237	8.6316	8.0000	2.9196	99.99%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(2)	1.8834	7.6316	7.0000	2.6583	99.97%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
BR$(3, \infty)$	2.1641	9.6316	9.0000	3.1809	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.8834	7.6316	7.0000	2.6583	99.97%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	2.0772	6.7368	6.0000	2.9579	99.90%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(2)	1.8729	5.7368	5.0000	2.5924	99.58%	99.98%	$\approx$100%	$\approx$100%	$\approx$100%
BR$(3, \infty)$	2.2815	7.7368	7.0000	3.3234	99.97%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.8729	5.7368	5.0000	2.5924	99.58%	99.98%	$\approx$100%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	1.5649	6.7368	6.0000	2.9088	98.05%	99.72%	99.93%	99.99%	$\approx$100%
DBR(2)	1.3961	5.9868	5.2500	2.6252	96.87%	99.50%	99.87%	99.98%	$\approx$100%
BR$(3, \infty)$	1.7337	7.7368	7.0000	3.4299	99.51%	99.97%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.5092	6.4868	5.7500	3.0408	99.22%	99.95%	99.99%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	1.0884	3.8461	2.9487	2.9054	96.40%	99.47%	99.87%	99.98%	99.99%
DBR(2)	0.9113	3.3497	2.4523	2.6160	95.05%	99.23%	99.81%	99.97%	99.99%
BR$(3, \infty)$	1.2654	4.8461	3.9487	3.6734	99.10%	99.95%	99.99%	$\approx$100%	$\approx$100%
DBR(3)	1.0253	4.1052	3.2078	3.2738	98.59%	99.91%	99.99%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	2.0070	9.6277	9.0515	2.9057	99.99%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(2)	1.8855	8.6279	8.0516	2.6775	99.97%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
BR$(3, \infty)$	2.1286	10.6277	10.0515	3.1341	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%
DBR(3)	1.8857	8.6285	8.0522	2.6779	99.97%	$\approx$100%	$\approx$100%	$\approx$100%	$\approx$100%

Coherence Protocol	Read Costs	Write Costs	Nbr of Copies	Op Costs	Data Availability $p = 0.75$	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 0.99$
BR$(2, \infty)$	1.1164	3.6456	2.7370	2.9130	96.34%	99.47%	99.87%	99.98%	99.99%
DBR(2)	0.9191	3.1426	2.2340	2.5943	94.68%	99.17%	99.79%	99.97%	99.99%
BR$(3, \infty)$	1.3136	4.6456	3.7370	3.7038	99.08%	99.95%	99.99%	$\approx$100%	$\approx$100%
DBR(3)	1.0625	3.9566	3.0480	3.2923	98.49%	99.90%	99.99%	$\approx$100%	$\approx$100%

Figure 22: *Comparison of static vs. dynamic Boundary-Restricted coherence protocols on the basis of different workloads*

In case of workload a), *grwr* varies between zero and eight, adopting each value for exactly the same amount of time (i.e. 11.1% of the application's run-time). Thus, write-intensive periods are as common as read-intensive periods. Additionally, in some periods, read requests and write requests are equally balanced. In workload case b), *grwr* is restricted to vary between zero and four, i.e. an application having such an behavior tends to write much more as in case a). The contrary is true for workload c). Here, periods of heavy read accesses to DSM are dominant. The same holds true for workload case d), but not as extreme as in the previous case. Case e) represents applications which either basically exclusively write to DSM 50% of the run-time or almost exclusively read from it in the remaining 50%. The workload cases f) and g) belong to applications with either write or read during almost the entire run-time. In contrast to the workloads b) and c), they do not show such a strict "all-or-nothing" behavior with respect to values adopted by *grwr* over time, but are more liberal. They cover the entire spectrum of *grwr*-values but in an ever decreasing (case f) or increasing manner (case g). Finally, workload case h), represents applications similar to the one given in Figure 21.

The analysis shows that the data availability offered by BR protocols and the corresponding DBR protocols, i.e. those having the same minimal number of cached copies in a write phase, are almost identical; they vary not more than about 0.5% for values of $p \geq 75\%$. This is in contrast to the mean operation costs; the DBR coherence protocols always incur lower costs. The savings range from 7.85% (workload g, $w = 2$) to as much as 22% (workload d, $w = 3$). (We used the mean operation costs of the BR protocols as base for this calculation). The overall result of this comparison is that *whenever* a minimal amount of data availability can be sacrificed (i.e. $\leq 0.5\%$), a DBR coherence protocol should be used instead of a static BR protocol. This results in a substantial decrease of operation costs and therefore improvement in application execution costs.

8 Conclusion

Since in large-scale networks failures are likely to arise, DSM systems must be scalable and fault-tolerant. In particular, the DSM coherence protocols should be able to continue service even when some participating sites fail or are not reachable due to communication breakdowns. In this paper, we extended a DSM coherence protocol class, namely the Boundary-Restricted coherence protocol class, in such a way, that the scalability and fault-tolerance properties inherent in instances of this class were applied to the new *Dynamic Boundary-Restricted* (DBR) coherence protocol class. A DSM server using a DBR coherence protocol dynamically monitors the current workload and adjusts certain protocol parameters, to guarantee a given minimal degree of DSM page data availability. Furthermore, the protocols assure that this is done at very low costs. For this purpose, the DSM server switches between protocol instances of the BR coherence protocol class. It always chooses an instance which achieves the data

availability degree given at the lowest costs possible under the currently given environmental conditions, i.e. the current workload. Analytical investigations showed that for various different workloads, a DBR coherence protocol outperforms a corresponding BR coherence protocol in terms of mean operation costs. Saving in the range from 7.85% to 22% were found. In terms of data availability, two comparable protocols behave almost identical: the DBR protocol showed at the most a data availability being 0.5% lower than its BR counterpart. Since the effort to gather and maintain statistics as well as performing dynamic changes of the protocol used, are very simple and inexpensive, DBR coherence protocols should be used over BR coherence protocols whenever a decrease of maximal 0.5% of data availability can be tolerated.

References

[1] M. Dubois and C. Scheurich. Memory access dependencies in shared-memory multiprocessors. *IEEE Transactions on Computers*, 16(6):660–673, June 1990.

[2] Elmootazbellah Nabil Ehnozahy, David B. Johnson, and Willy Zwaenepoel. The performance of consistent checkpointing. In *Proceedings of the* 11^{th} *Symposium on Reliable Distributed Systems*, pages 39–47, Houston, Texas, USA, 1992. IEEE-CS, IEEE-CS Press.

[3] Michael J. Feeley, Jeffrey S. Chase, Vivek R. Narasayya, and Henry M. Levy. Integrating coherency and recoverability in distributed systems. In *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI'94)*, Monterey, CA, USA, November 1994. USENIX, ACM SIGOPS, USENIX.

[4] Brett D. Fleisch, Randall L. Hyde, and Niels Christian Juul. MIRAGE+: A Kernel Implementation of Distributed Shared Memory on a Network of Personal Computers. *Software—Practice & Experience*, 24(10):887–910, October 1994.

[5] Brett D. Fleisch and Gerald J. Popek. Mirage: A coherent distributed shared memory design. In Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, published in *Operating Systems Review* 23(5) Special Issue, pages 211–223, The Wigwam, Litchfield Park, Arizona, USA, December 1989. ACM SIGOPS, ACM Press.

[6] J.R. Goodman. Cache consistency and sequential consistency. Technical report 61, SCI Committee, March 1989.

[7] B. Janssens and W. K. Fuchs. Relaxing consistency in recoverable distributed shared memory. In *Proceedings of the Twenty-Third Annual International Symposium on Fault-Tolerant Computing: Digest of Papers*, pages 155–163, June 1994.

[8] N. C. Juul and B. D. Fleisch. A Framework for Consistency and Recoverability in Distributed Shared Memory. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, September 1995. Submitted for publication.

[9] N. C. Juul and B. D. Fleisch. A Memory Approach to Consistent, Reliable Distributed Shared Memory. In *Proceedings of the 5th Symposium on Hot Topics in Operation Systems*, May 1995. To be published.

[10] Richard Koo and Sam Toueg. Checkpointing and roolback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, SE-13(1):23–31, January 1987.

[11] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, September 1979.

[12] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[13] Virginia Lo. Operating systems implementations of distributed shared memory. *Advances in Computers*, 39, 1994.

[14] D. D. E. Long, A. Muir, and R. Golding. A Longitudinal Survey of Internet Host Reliability. Technical Report UCSC–CRL–95–16, Dept. of Computer and Information Sciences, University of California, Santa Cruz, Santa Cruz, CA 95064, March 1995.

[15] Ajay Mohindra and Umakishore Ramachandran. A survey of distrbuted shared memory in loosely-coupled systems. Technical Report GIT-CC-91/01, Georgia Institute of Technology, Atlanta, GA, USA, January 1991.

[16] N. Neves, M. Castro, and P. Guedes. A checkpoint protocol for an entry consistent shared memory system. In *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing (PODC'94)*. ACM, ACM Press, August 1994.

[17] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52–60, August 1991.

[18] James S. Plank and Kai Li. ickp: A consistent checkpointer for multicomputers. *IEEE Parallel & Distributed Technology*, 2(2):62–67, Summer 1994.

[19] Golden G. Richard, III and Mukesh Singhal. Using logging and asynchronous checkpointing to implement recoverable distributed shared memory. In *Proceedings of the* 12^{th} *Symposium on Reliable Distributed Systems*, pages 86–95, Princeton, New Jersey, USA, October 1993. IEEE-CS, IEEE-CS Press.

[20] M. Satyanarayanan, Henry H. Mashburn, Puneet Kumar, David C. Steere, and James J. Kistler. Lightweight recoverable virtual memory. *ACM Transactions on Computer Systems*, 12(1):33–57, February 1994.

[21] A. Silberschatz and P. B. Galvin. *Operation System Concepts.* Addison-Wesley Publishing Company, Inc., 4th edition edition, 1994. ISBN 0-201-50480-4.

[22] M. Singhal and N. G. Shivaratri. *Advanced Concepts in Operation Systems: distributed, database, and multiprocessor operating systems.* McGraw-Hill, Inc., 1994. ISBN 0-07-057572-X.

[23] M. Stumm and S. Zhou. Fault tolerant distributed shared memory algorithms. In *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*, pages 719–724. IEEE-CS, IEEE Press, December 1990.

[24] O. E. Theel and B. D. Fleisch. Design and Analysis of Highly Available and Scalable Coherence Protocols for Distributed Shared Memory Systems Using Stochastic Modeling. Technical Report UCR-CS-95-1, University of California at Riverside, Dept. of Computer Science, April 1995.

[25] Kun-Lung Wu and W. Kent Fuchs. Recoverable Distributed Shared Virtual Memory. *IEEE Transactions on Computers*, 39(4):460–469, April 1990.