To appear in the Proceedings of 12th ACM Symposium on Parallel Algorithms and Architectures, July 2000.

Comparing the Effectiveness of Fine-Grain Memory Caching against Page Migration/Replication in Reducing Traffic in DSM Clusters

An-Chow Lai and Babak Falsafi
School of Electrical & Computer Engineering
Purdue University
1285 EE Building
West Lafayette, IN 47907
impetus@ecn.purdue.edu
http://www.ece.purdue.edu/~impetus

Abstract

In this paper, we compare and contrast two techniques to improve capacity/conflict miss traffic in CC-NUMA DSM clusters. Page migration/replication optimizes read-write accesses to a page used by a single processor by migrating the page to that processor and replicates all read-shared pages in the sharers' local memories. R-NUMA optimizes read-write accesses to any page by allowing a processor to cache that page in its main memory. Page migration/replication requires less hardware complexity as compared to R-NUMA, but has limited applicability and incurs much higher overheads even with tuned hardware/software support.

In this paper, we compare and contrast page migration/replication and R-NUMA on simulated clusters of symmetric multiprocessors executing shared-memory applications. Our results show that: (1) both page migration/replication and R-NUMA significantly improve the system performance over "first-touch" migration in many applications, (2) page migration/replication has limited opportunity and can not eliminate all the capacity/conflict misses even with fast hardware support and unlimited amount of memory, (3) R-NUMA always performs best given a page cache large enough to fit an application's primary working set and subsumes page migration/replication, (4) R-NUMA benefits more from hardware support to accelerate page operations than page migration/replication, and (5) integrating page migration/replication into R-NUMA to help reduce the hardware cost requires sophisticated mechanisms and policies to select candidates for page migration/replication.

1 Introduction

Clusters of symmetric multiprocessors (or SMPs) have emerged as the architecture of choice for building medium- to large-scale parallel servers. To preserve software compatibility and portability with respect to SMPs, designers often connect a cluster of SMPs using a high-bandwidth/low-latency switch-based network and a directory-based distributed shared-memory (DSM) protocol. DSM provides a shared global address space over SMPs' physically distributed memory. Despite a compatible programming interface, performance tuning applications on DSMs is often difficult because remote shared-memory accesses inherently take up to ten to a hundred times longer than local memory accesses.

To reduce remote memory traffic, most DSM clusters use a Cache-Coherent Non-Uniform Memory Access (CC-NUMA) architecture to cache remote data in both processor caches and specialized cluster caches on every node and exploit memory access locality [11,14]. Recent designs for aggressive remote caching propose incorporating dedicated cluster caches into the DSM hardware to cache remote data [14,21]. Unfortunately, while remote caching in CC-NUMA substantially removes accesses to remote memory in most workloads, many scientific and commercial applications still exhibit high capacity/conflict misses in the cache hierarchy and result in significant remote memory traffic [12,1].

To address this problem, recent DSMs [6,10] incorporate a number of techniques to reduce capacity/conflict traffic in CC-NUMA. One approach to reduce capacity/conflict memory traffic in CC-NUMA is to use kernel-based page migration/replication to improve data locality on every node [18,10]. Page migration/replication dynamically monitors the system-wide memory access frequency to a shared page and either migrates the page to the memory of the page's most frequent user, or replicates the page in all the sharers' main memory when the page is mostly read-shared. Migrating/replicating a page on a node converts remote memory accesses to local accesses on that node, thereby reducing remote traffic.

Page migration/replication, however, only reduces traffic for read-write memory pages that are primarily accessed by a single DSM node for a long period of time or read-shared memory pages. As such, page migration/replication does not benefit memory pages that are actively shared by multiple DSM nodes and incur high capacity/conflict misses in CC-NUMA. Moreover, page migration/replication requires global coordination among the DSM nodes to inform a page's sharers that either the home node location (in the case of migration) or access protection (in the case of replication) for the page is changing. Such operations take over tens of microseconds even in systems with hardware support for page invalidation and movement — e.g., SGI Origin2000 [10] — significantly limiting the applicability of page migration/replication and diminishing the opportunity to reduce traffic.

Alternatively, other DSMs incorporate aggressive hardware-intensive techniques to implement fine-grain (remote) memory caching [8,4,5,16] — e.g., Sun WildFire [6]. These designs are based on integrating the base CC-NUMA protocol with Simple COMA [7], enabling a processor to store coherent remote memory blocks in main memory pages. Because main memory provides a much larger repository for remote caching, these designs potentially eliminate the capacity/conflict traffic in DSM while obviating the need for cluster caches in the DSM hardware. One such proposal for fine-grain memory caching is Reactive NUMA (R-NUMA) [5] in which hardware on every node monitors the capacity/conflict activity for remote data and dynamically selects between the CC-NUMA and S-COMA protocols on a per-page basis. By placing

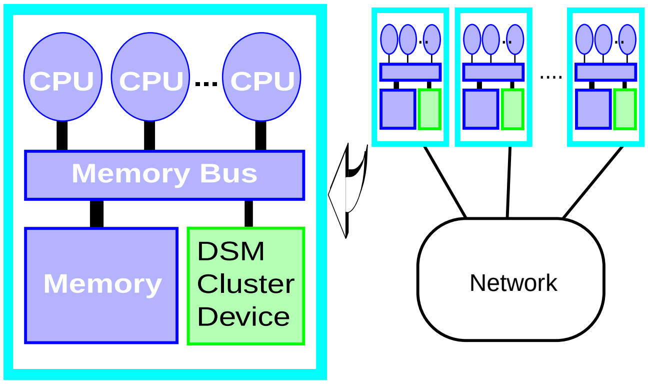

FIGURE 1. A DSM cluster.

data that often incur capacity/conflict misses in CC-NUMA in main memory, R-NUMA significantly reduces remote traffic.

R-NUMA only requires local coordination because every node selects and implements a caching policy for a page independent of the others. R-NUMA incurs much lower overheads moving pages than page migration/replication because it only requires flushing a single page, shooting down TLBs on a single node, and retrieving only the necessary set of remote blocks on the page. R-NUMA's ability for page caching, however, is limited because practical implementations limit remote caching to only a fraction of main memory due to the extra fine-grain tag hardware overhead to implement cache block-level coherence. Sparse memory access patterns also limit R-NUMA's performance due to page fragmentation which significantly increases demand on main memory page allocation and incurs high overhead.

This paper compares and contrasts fine-grain memory caching and page migration/replication as a technique to reduce capacity/conflict memory traffic in CC-NUMA. In this paper, we focus on reducing capacity/conflict traffic on *data* pages in single parallel applications. Page migration/replication has been shown to be quite effective for multiprogrammed workloads and instruction pages [17,18]. We evaluate the effectiveness of page migration/replication and R-NUMA in reducing capacity/conflict traffic in DSM clusters by executing shared-memory applications on simulated systems. Our results indicate that:

1. Both page migration/replication and R-NUMA substantially improve the system performance over ‘‘first-touch’’ migration in many applications.
2. Page migration/replication has limited opportunity and can not eliminate all the capacity/conflict misses even with fast hardware support and unlimited amount of memory, improving performance by 20% on average over CC-NUMA; data page replication is applicable and reduces misses substantially in only one out of seven applications, and page migration is infrequent due to both high read-write sharing degrees and static sharing behavior of data pages.
3. R-NUMA always performs best given a large enough page cache to fit an application's primary working set, subsuming page migration/replication, and improving performance over CC-NUMA by 40%; R-NUMA simply allocates and places read-write sharing pages into the page cache eliminating the capacity/conflict misses.

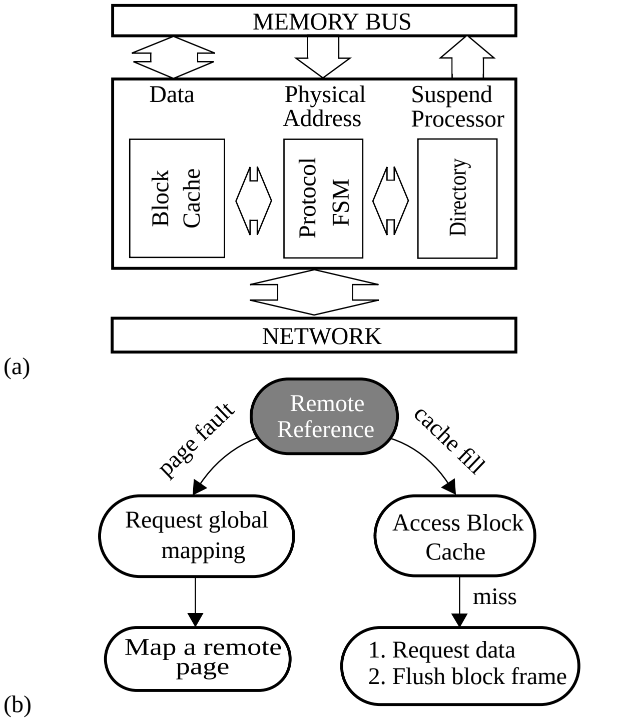

FIGURE 2. Remote caching in CC-NUMA: (a) a CC-NUMA cluster device, and (b) action sequence on a remote miss.

4. R-NUMA is much more sensitive to page operation overhead and benefits more from fast support for page invalidation and movement (e.g., page flushing and TLB shootdowns) than page migration/replication due to R-NUMA's much higher frequency of page operations.
5. Integrating page migration/replication into R-NUMA to help reduce the hardware cost requires sophisticated mechanisms and policies to select candidates for page migration/replication; relocating pages into R-NUMA's page cache interferes with accurate readings of miss counters for page migration/replication limiting the latter's opportunity to get invoked.

The next section describes the basic CC-NUMA distributed shared-memory machine structure we study in this paper. Section 3 provides more details of our DSM designs with page migration/replication and fine-grain memory caching support. Section 4 presents a qualitative analysis of the performance of the various systems we study in this paper. Section 5 describes the simulation methodology we use to evaluate system performance. Section 6 presents the results of our simulations. Finally, Section 7 and Section 8 discuss the related works and conclude the paper respectively.

2 Base CC-NUMA DSM Cluster

Figure 1 illustrates the basic distributed shared-memory cluster organization that we study in this paper. Sun's WildFire cluster [6], Fujitsu's Sinfinity NUMA [19], Data-General's NUMALiiNE [3], and Sequent's STiNG [12] are all examples of such DSM clusters. Each node is a symmetric multiprocessor (SMP) workstation with four processors connected via a coherent bus to an interleaved memory. A DSM cluster device implements a directory-based cache coherence protocol to extend the shared-memory abstraction

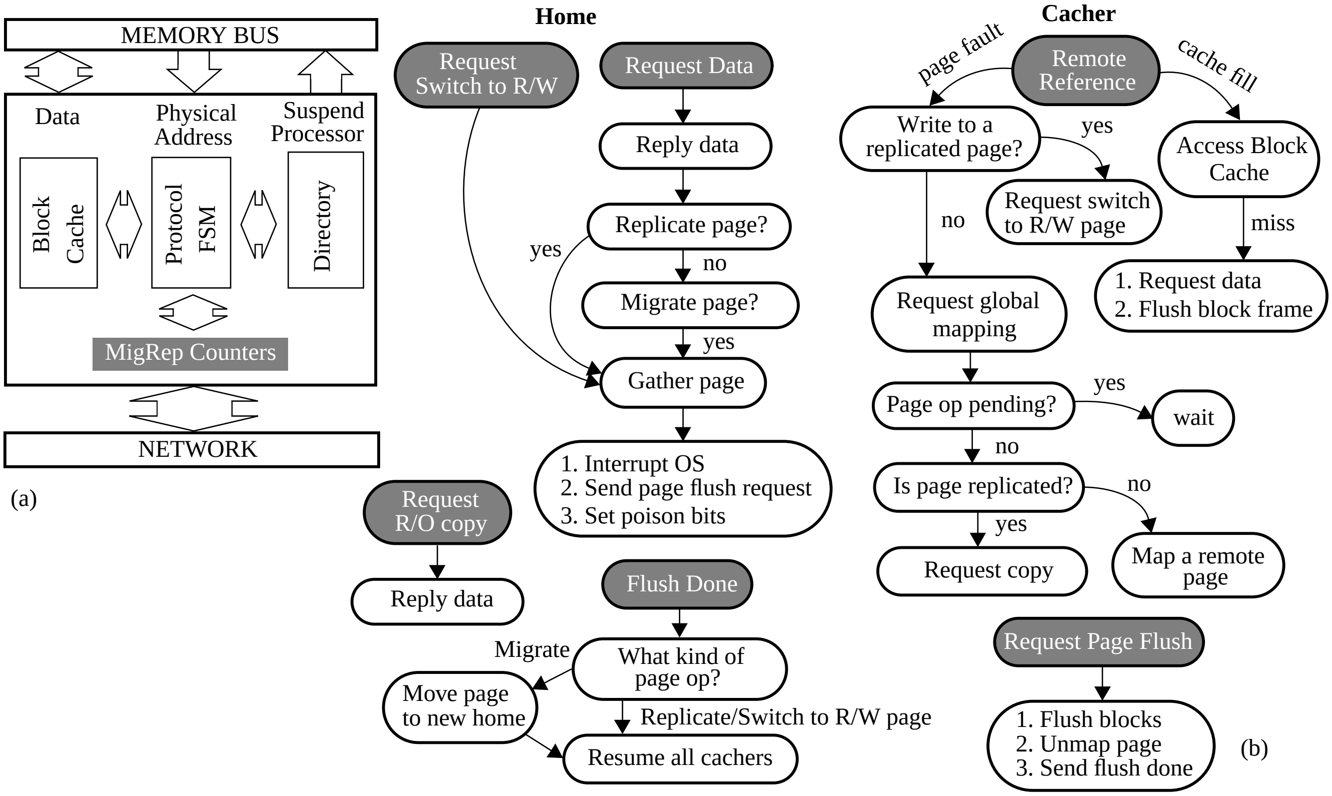

FIGURE 3. CC-NUMA+MigRep: (a) a CC-NUMA+MigRep cluster device, (b) sequence of actions at home and at cacher.

across the nodes. This device implements the same basic coherence protocol in all systems. For the systems we study, the device differs in the necessary hardware support for caching and page operations as required by each system (as described below and in Section 3).

CC-NUMA forms the basis of comparison among the systems we study. Most distributed shared-memory clusters [19,3,12,11] are CC-NUMA machines. Figure 2 (a) illustrates the anatomy of a DSM cluster device for these machines. The device is equipped with an cluster cache (also known as remote cache [21] or block cache [14]) that holds recently referenced remote data blocks. To differentiate this cache from the page-granularity caches in Simple COMA (S-COMA) [7] and Reactive NUMA (R-NUMA) [5], we will refer to it as the *block cache*. A directory maintains the sharing status of all the blocks residing in the node's main memory. A hardware finite-state machine implementation of the coherence protocol manages accesses to the directory and the block cache, services messages from the remote nodes, and requests remote data on behalf of the node.

Figure 2 (b) illustrates the flow of events on a remote reference in CC-NUMA. All initial accesses to remote data result in a (soft) page fault. The node's operating system software requests and receives the page's global mapping consisting of a home node id and a physical page address. The operating system maps the page accordingly, updates the node's page tables, and resumes the faulting processor. Subsequent references to a mapped page result in cache block fill requests on the SMP memory bus. The cluster device satisfies the cache fill requests for remote data out of the block cache by snooping for physical addresses on the memory bus. Upon a miss in the block cache, the cluster device allocates a block frame in the block cache, replacing and writing back dirty blocks if necessary, and invokes the coherence protocol to fetch the remote data.

In this paper, we only consider fast and small SRAM-based block caches. Alternatively, some designs incorporate large but slow DRAM-based block caches [17,2,21]. The latter reduce the capacity/conflict miss traffic in CC-NUMA at the cost of increasing the cache look up time and the controller occupancy. To keep the latencies and occupancies comparable among the block-cache and page-cache based systems in this study, we only consider SRAM-based block caches. A detailed study of the block cache design space is beyond the scope of this paper and has been dealt with in great detail in a recent paper [14].

Prior research indicates that CC-NUMA's performance may be very sensitive to the initial data allocation and placement [9]. As such, in this paper we use a first-touch placement policy in *all* the systems we study. This policy is simple and has been shown to substantially eliminate unnecessary traffic [13]. In this policy, an user-invoked directive on every node initiates page migration and placement at the start of the parallel phase of the program. Upon the first request for each page, the home node migrates the page to the requester, assuming the first requester is likely to prove a frequent requester. This is especially true for some regular scientific applications that specifically "touch" pages to ensure their proper placement [20].

3 Reducing Capacity/Conflict Traffic

We evaluate two techniques to reduce the capacity/conflict traffic in CC-NUMA: (1) page migration/replication (MigRep) used in SGI Origin 2000 [10], and (2) R-NUMA enabling selective fine-grain caching of remote data in local memory used in Sun Wild-

Fire [6]. In this section, we describe how each technique works and present the required cluster device hardware support. Section 4 presents a qualitative performance analysis of the two techniques. Section 6 presents simulation results of shared-memory applications comparing the two techniques..

3.1 CC-NUMA+MigRep

Page migration/replication reduces the capacity/conflict traffic in CC-NUMA by migrating a page to the memory of the page's most frequent user, and replicating the mostly read-shared pages in sharers' local memories. In this paper, we evaluate page migration/replication as a technique to reduce traffic for *data* pages in the context of single parallel programs. Page migration/replication has also been shown to be quite effective in reducing traffic for both code and data pages in multiprogrammed environments of sequential and parallel jobs [18].

Figure 3 (a) illustrates the cluster device for a CC-NUMA with page migration/replication hardware support (CC-NUMA+MigRep). Page migration/replication slightly differs from the base CC-NUMA system in that it includes page reference monitoring hardware, i.e. per-page per-node miss rate counters to detect candidates for page migration/replication.

Figure 3 (b) illustrates the flow of events for page migration/replication both on the home node and on the cachers. Upon receiving a request for a cache fill from a remote node, the home node increments the appropriate page miss counter and checks if a page replication or migration is necessary. The hardware compares the miss counters against a preset threshold to decide if a page operation is necessary. In the case of replication, if the write miss counters are zero and the read miss counters from the requesting node are greater than the threshold, a page replication is invoked. In the case of migration, if the requester's miss counters are greater than the home's by at least the threshold value, a page migration is invoked. The miss counters are reset periodically at a preset interval.

Upon a page replication/migration, the hardware invokes a soft trap and the operating system begins to migrate or replicate the page. A request for a replicated page copy at home simply results in a reply with the appropriate data. A write protection fault to a replicated page by a cacher results in a request at the home node to switch the page back to a read-write page. Both page migration/replication and requests to switch a page to a read-write mode invoke a soft trap at the home node's operating system to perform a *page invalidation and data gathering* operation.

Page gathering requires locking the page mapper, gathering the page from all the sharers, setting the poison bits [10] for all the blocks on the page (to allow lazy TLB invalidation), moving the page to the new home, and shooting down the home TLB. Upon receiving a page flush request, the DSM hardware invalidates and flushes all the cache blocks for the page. In systems with no hardware support for a page flush and lazy TLB invalidation, flushing a page incurs much higher overheads because the message arrival first invokes a software trap for the kernel taking over the job of flushing the blocks, and shooting down the TLB [18]. Once the page is flushed, the kernel on the home node will unlock the page mapper, move the page to a new home in the case of migration, and resume all cachers waiting for the page operation to complete. Much like page flushing, page copying can be accelerated using hardware [10]. In this paper, we study the system's performance sensitivity to page migration/replication's speed.

Upon a page fault at the cachers, the kernel checks if the fault is an unmapped page or a protection fault. If it is an unmapped page fault to a replicated page, the kernel simply requests a page copy from the home node. Otherwise, the page is mapped as in CC-NUMA. Upon a protection fault, the kernel requests the home node to switch the page back to a read-write cached page.

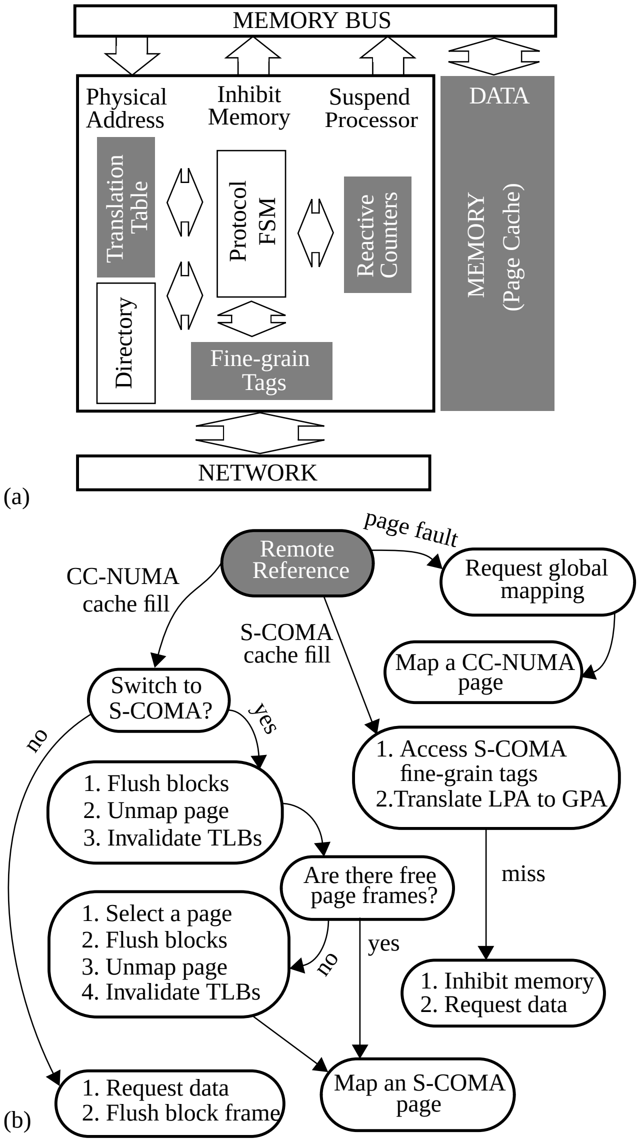

FIGURE 4. Remote caching in R-NUMA: (a) an R-NUMA cluster device, and (b) action sequence on a remote miss.

3.2 R-NUMA

Reactive NUMA (R-NUMA) [5] is a hybrid DSM architecture which adaptively switches the per-page caching policy between CC-NUMA and Simple COMA (S-COMA) [7]. S-COMA enables the system to allocate main memory page frames to place remote data while managing coherence and sharing at cache block granularity. S-COMA allows a DSM to take advantage of the large capacity of a node's main memory to store remote data. S-COMA, however, significantly increases the hardware complexity as compared to CC-NUMA by requiring: (1) fine-grain block sharing tags to enforce coherence for cached data in main memory, and (2) hardware to translate local main memory addresses to a global shared address to locate the home node when references miss in

Mechanism	Read-Only Miss Reduction	Read/Write Miss Reduction		Page Operation Overhead	Page Operation Frequency
		Low Sharing Degree	High Sharing Degree		
Page Replication	yes	no	no	high	low
Page Migration	no	yes	no	high	low
R-NUMA	yes	yes	yes	low	much higher

TABLE 1. Capacity/conflict miss reduction opportunity and overhead.

the memory cache. R-NUMA, however, obviates the need to implement a block cache in the cluster device [5,14].

Figure 4 (a) illustrates the cluster device for R-NUMA machines. R-NUMA includes the base CC-NUMA DSM hardware, the S-COMA fine-grain tags and reverse translation table, and page miss rate counters. Figure 4 (b) illustrates the flow of events for caching remote data in R-NUMA. The operating system initially maps the page CC-NUMA. R-NUMA captures the behavior of program by using a per-page per-node *refetch* counter to count the number of times the hardware fetches a block recently cached but replaced due to capacity/conflict traffic in the processor's cache. When the counter exceeds a pre-defined threshold, the requesting processor generates an interrupt to the operating system to remap the CC-NUMA page into a local S-COMA page so that future misses to the page can be satisfied in the processor's local memory. Since the remapping is a local page operation, it does not affect other processors' decisions. Cache fills for the (S-COMA) page-cache blocks require access to the S-COMA tags to check for coherence and to translate a local physical address (LPA) to a global physical address (GPA). If the block is not in the page cache, the cluster device inhibits the fill and requests the corresponding home node for the remote block.

4 Qualitative Performance Analysis

Both CC-NUMA+MigRep and R-NUMA have their advantages and disadvantages. In this section, we qualitatively evaluate the trade-offs between the two systems in terms of the opportunity for reducing capacity/conflict traffic, and the runtime overheads associated with both selecting a candidate page to perform an operation on (e.g., move to page cache or migrate) and actually performing the page operation (Table 1).

4.1 Opportunity to Reduce Misses

Page migration/replication's performance highly depends on an application's sharing characteristics and the resulting opportunity for reducing remote misses. Page migration works best when a page's read-write sharing degree is low but the page miss rate is high. When the sharing degree is low, for example in the case of a single frequent reader and/or writer, migrating the page eliminates the remote misses to the page and substantially reduces the traffic. On the other hand, when the sharing degree is high, eliminating the remote misses at one sharer does not eliminate the remote misses at others and therefore page migration does not benefit pages with several simultaneous sharers. Page migration also helps pages with dynamically varying list of sharers where the home node no longer shares the page. For such pages, page migration dynamically reacts to the change in the sharer's list and always makes sure that one of the active sharers becomes the home node for the page. Page replication works best for pages that are read-shared for a long time, and does not benefit pages with high write frequency.

R-NUMA works for all pages with high rate of capacity/conflict misses including pages with high read-write sharing degree. Because it allows read-write caching of data in any sharer's local memory, R-NUMA can reduce the overall number of capacity/conflict misses in the system. The opportunity for reducing capacity/conflict misses in R-NUMA, however, highly depends on memory pressure on the node [8,5]. R-NUMA can suffer from the problem of page fragmentation and requires large amount of memory to meet the working sets of applications. Practical implementations of R-NUMA [6] also limit the page cache size to reduce hardware complexity and cost. Therefore, high memory pressure in some applications may result in frequent page deallocation from the page cache, diminishing the opportunity for reducing capacity/conflict misses.

In the limit, with a large enough page cache, R-NUMA can eliminate all the capacity/conflict traffic in the system. In contrast, page migration/replication given enough memory can at best eliminate read-write traffic from one sharer and all the read-only traffic. .

4.2 Page Selection/Operation Overhead

Page migration/replication at a minimum incurs the high overhead of page invalidation and data gathering. The latter at a minimum incurs the overhead of taking a soft trap, flushing the page at all cachers, and moving or copying the page. Such an operation requires global coordination and is slow even in systems with hardware support for page migration/replication [10], taking tens of thousands of processor cycles. In systems with no hardware support, the operation invokes the kernel on every node to invalidate the pages [17] and immediately shoots down the TLBs. Moreover, with no support for block copying hardware, the kernel on the home node must send the page through the DSM board one block at a time. The resulting page migration/replication in systems with no hardware support, incurs an order of magnitude more overhead [18].

Because page migration/replication incurs the high overhead of page gathering, it requires a long page selection interval to make accurate decisions about selecting a candidate page for migration/replication. Such a constraint significantly limits the frequency at which a page can be migrated/replicated, thereby diminishing the overall opportunity for miss reduction even in systems with hardware support for page migration/replication.

In contrast, in R-NUMA the decision as to whether to cache remote data using CC-NUMA or S-COMA is an entirely local one and needs not involve other cachers. It only requires flushing a single page, shooting down TLBs on a single node, and refetching only the necessary cache blocks. Such a local operation incurs less overhead, can be overlapped with computation on other processors/nodes, and does not require sophisticated page copying hardware. However, in the presence of memory pressure in the page

Application	Problem	Input Data Set
barnes	Barnes-Hut N-body simulation	16K particles
cholesky	Blocked sparse Cholesky factorization	tk16.O
fmm	Fast Multipole N-body simulation	16K particles
lu	Blocked dense LU factorization	512x512 matrix, 16x16 blocks
ocean	Ocean simulation	130x130 ocean
radix	Integer radix sort	1M integers, radix 1024
raytrace	3-D scene rendering using ray-tracing	car

TABLE 2. Applications and input parameters.

cache, R-NUMA can result in a high page allocation/deallocation frequency, increasing the overall overhead incurred in page operations.

5 Methodology

To compare practical implementations of CC-NUMA, CC-NUMA+MigRep, R-NUMA, we use the Wisconsin Wind Tunnel II [15] to simulate a distributed shared-memory machine consisting of a network of eight SMP nodes (Figure 1). Each node is a 4-way multiprocessor with 600 MHz dual-issue processors interconnected by a 100 MHz split-transaction bus. A snoopy MOESI coherence protocol keeps the caches within each node consistent. We assume perfect instruction caches[1] but model data caches and their contention at the memory bus accurately. We further assume a point-to-point network with a constant latency of 80 cycles but model contention at the network interfaces accurately.

Table 2 presents the applications we use in this study and the corresponding input parameters. *Barnes*, *cholesky*, *fmm*, *lu*, *ocean*, *radix* and *raytrace* are from the SPLASH-2 [20] benchmark suite.

Table 3 presents the costs of block and page operations in processor cycles for our base system assumptions. SRAM devices include the block cache, S-COMA fine-grain tags and translation table, R-NUMA reactive counters, and CC-NUMA+MigRep miss counters. DRAM accesses correspond to accesses to the page cache. Soft traps include page faults and R-NUMA relocation interrupts. Page allocation/replacement involves taking a soft trap, invalidating the (local) TLBs, and flushing the blocks back to the home node. The overhead varies depending on the number of blocks flushed. Page relocation in R-NUMA uses similar mechanisms as page allocation/replacement and incurs the same overheads.

Our application data set sizes are selected to be small enough so as not to require prohibitive simulation cycles, while being large enough to maintain the intrinsic communication and computation characteristics of the parallel application. Woo, et al., characterize the behavior of SPLASH-2 applications in terms of working sets and show that for most of the applications, the data sets provided have a primary working set that fits in an 8-Kbyte cache [20]. We, therefore, conservatively assume 16-Kbyte (direct-mapped) processor caches to compensate for the small size of the data sets.

Operation	Cost(processor cycles)
block operations	
Network latency	80
Local miss latency	104
Round-trip remote miss latency	418
page operations	
soft traps	3000
TLB shootdown	300
allocation/replacement or R-NUMA relocation	3000~11500
migration/replication operations	
page invalidation and data gathering	3000~11500
page copying	8000~21800

TABLE 3. Base line system assumptions.

To facilitate integrating the node's SMP protocol (derived from SPARC) with the simple DSM write-invalidate protocol and to eliminate race conditions when resuming processors waiting for a remote block, we assume CC-NUMA block caches that maintain inclusion with the node's processor caches [6]. As such, we simulate block caches equal in size to the sum of all the processor cache sizes. This assumption helps mitigate any adverse effects due to the inclusion requirement of read-write blocks. Consequently, a four-processor node will have a 64-Kbyte CC-NUMA block cache. To compensate for the lower cost of DRAM as compared to SRAM, our base system assumes an S-COMA page cache of 2.4 Mbytes, a factor of 40 larger than our CC-NUMA block cache.

Unless specified otherwise, all experiments assume a lazy TLB shootdown strategy for page migration/replication using directory poisoning and page copying hardware, characteristic of aggressive systems with hardware/software support for page migration/replication [10]. Our system provides hardware support for a page flush during data gathering, obviating the need for soft traps upon page invalidation. We use page migration/replication threshold of 800 misses, a reset interval of 32000 misses, and an R-NUMA switching threshold of 32 misses across all the benchmarks. The threshold values are selected so as to optimize performance over all benchmarks. To gauge page migration/replication's best performance, we assume no memory pressure for CC-NUMA+MigRep in any of the experiments — i.e., our system assumes that a free page in memory is always available for the purpose of migration/replication.

6 Results

In this section, we present results from our simulation experiments. We first show base system performance results. Next, we present the performance sensitivity of CC-NUMA+MigRep and R-NUMA to slow hardware/software support for page operations. Third, we study the impact of long network latency on the performance of the systems. Finally, we investigate whether page migration/replication can help reduce the hardware requirements in R-NUMA. All

[1] The scientific codes we study have low instruction cache miss ratios. This assumption may not hold for all applications.

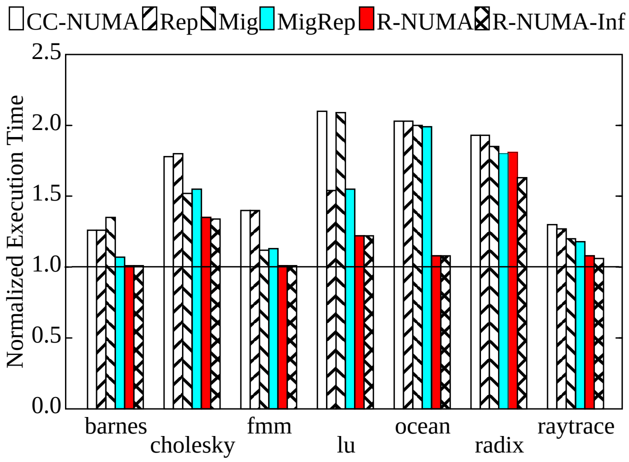

FIGURE 5. Base performance comparison for CC-NUMA, CC-NUMA+MigRep, and R-NUMA.

execution times are normalized against a perfect CC-NUMA — i.e. CC-NUMA with an infinite block cache — unless specified otherwise.

6.1 Opportunity to Reduce Misses

Figure 5 plots the performance of our base CC-NUMA, a CC-NUMA with page replication (Rep), a CC-NUMA with page migration (Mig), a CC-NUMA with both page migration/replication (MigRep), R-NUMA, and an R-NUMA with an infinite page cache (R-NUMA-Inf). All numbers are normalized against perfect CC-NUMA. Not surprisingly, capacity/conflict traffic significantly impacts execution time in CC-NUMA increasing execution times to 60% over perfect CC-NUMA.

The figure indicates that data page migration/replication can significantly improve the performance by 20% on average over CC-NUMA after "first-touch" migration. Page migration/replication, however, suffers from limited opportunity and fails to eliminate capacity/conflict misses. R-NUMA, however, performs best and improves execution time in CC-NUMA by 40% on average. R-NUMA only suffers from high page relocation overhead in two applications, *cholesky* and *radix*. In the limit, R-NUMA-Inf is as good as perfect CC-NUMA in two benchmarks and is close to perfect CC-NUMA in another two benchmarks. The figure also shows that the applications vary in demand for page migration/replication. Five applications benefit from page migration, one application benefits from page replication, and one application does not need benefit from either.

Table 4 depicts the per-node number of page operations (i.e., page migration/replication in CC-NUMA+MigRep and page relocation in R-NUMA) in the systems. The table also depicts the breakdown of the per-node overall number of misses and the number of capacity/conflict misses (in parenthesis) in CC-NUMA, CC-NUMA+MigRep, and R-NUMA. The table corroborates the results in Figure 5 that page migration/replication occurs infrequently in many applications and as such does suffers from lack of opportunity to reduce conflict/capacity misses. For most of the applications, R-NUMA has more opportunities to reduce the number of remote misses because it has higher page operation frequency than page migration/replication.

Although there are a lot of page replications in *barnes* and *cholesky*, many of them are not on the execution's critical path and some of them are incorrect decisions. In contrast, *fmm, ocean,* and *radix* do not require page replication at all. In *barnes*, when only page migration is used, the performance gets worse because page migration unnecessarily migrates some of the read-only pages (Figure 5). However, adding page replication to page migration helps identify and remove the pages to be replicated so that page migration selects the appropriate pages and reduces capacity/conflict traffic. *Radix* and *raytrace* exhibit the same positive collaboration between page migration and page replication but the overall effect on execution time is smaller. In *cholesky* and *fmm*, performance of page migration/replication comes directly from page migration through improving data locality. Low reuse of migrated/replicated pages limits the performance improvement in *cholesky* and *raytrace*. *Lu* does not benefit from page migration but exhibits high benefits from page replication due to a read phase of reading the matrix to be factorized before the start of computation in each iteration. In *ocean* and *radix*, there are only a few candidates for page migration/replication.

Benchmarks	Number of Page Operations			Number of Overall Misses (Capacity/Conflict Misses) in x1000		
	Migration	Replication	Page Cache Relocation	CC-NUMA	CC-NUMA+MigRep	R-NUMA
barnes	9	133	19	1210 (1171)	159 (120)	39 (0)
cholesky	75	430	777	262 (169)	175 (82)	180 (88)
fmm	54	6	156	267 (221)	214 (168)	54 (8)
lu	135	167	417	1331 (1287)	376 (332)	73 (29)
ocean	37	0	201	300 (209)	267 (176)	104 (13)
radix	1	0	1714	157 (111)	153 (107)	100 (75)
raytrace	5	283	1059	597 (446)	257 (106)	213 (72)

TABLE 4. Number of per-node page operations and remote misses in CC-NUMA, CC-NUMA+MigRep, and R-NUMA.

R-NUMA virtually eliminates the capacity/conflict misses in all applications except for *cholesky*, *radix*, and *raytrace*. *Cholesky* and *radix* are kernels and as such do not exhibit reuse of the pages relocated into R-NUMA's page cache. Every relocation, however, requires refetching the flushed blocks which incurs a large number of misses in these applications. R-NUMA's performance in *radix* also suffers slightly from limited page cache capacity as shown by the improvement in performance from R-NUMA-Inf. Early relocation (i.e., through smaller threshold values) would significantly help these applications. In *raytrace*, the capacity/conflict misses remaining in R-NUMA are not on the execution's critical path and as such do not affect execution time.

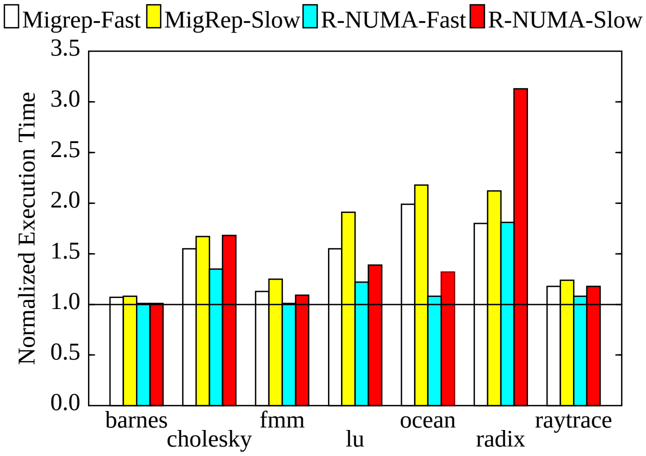

FIGURE 6. Performance comprison of CC-NUMA+MigRep and R-NUMA for systems with fast and slow page operation support.

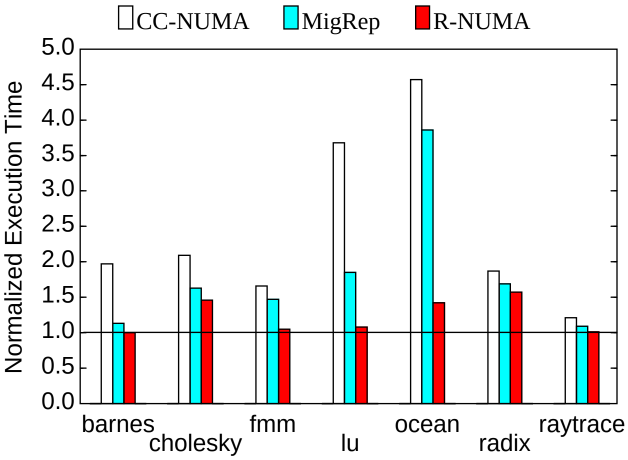

FIGURE 7. Performance comparison of CC-NUMA, CC-NUMA+MigRep, and R-NUMA for remote miss latencies four times larger than the base system.

6.2 Sensitivity to Page Operation Overhead

Conventional page migration/replication proposals relied on kernel-based solutions with no hardware/software for page operations. Recent studies indicate that commodity operating systems are not tuned for page migration/replication incurring prohibitively high overheads [24]. In this section, we present results evaluating the impact of slow page operations on the performance of page migration/replication and R-NUMA.

We assume an increase of ten-fold in page operation overheads as compared to the base systems (Table 3). Our slow system (as compared to the fast base system) assumes 50 µs (or 30000 cycles) for soft traps, 5 µs (or 3000 cycles) for TLB shootdown, and an additional page copying overhead of 10 µs (or 6000 cycles) per page. A larger page operation overhead also requires larger threshold values to prevent page operation frequency to increase to prohibitive levels resulting in page thrashing. Our slow systems use a threshold value of 1200 misses for CC-NUMA+MigRep and 64 misses for R-NUMA respectively.

Figure 6 compares the performance of CC-NUMA+MigRep and R-NUMA for systems with fast and slow page operations. These execution times are normalized against the perfect CC-NUMA. Our results indicate that on average page migration/replication is less sensitive to page operation overhead than R-NUMA. This result follows from Table 4, which indicates that our applications exhibit much lower page migration/replication frequency in CC-NUMA+MigRep than page relocation in R-NUMA. Although R-NUMA's per-page overhead is lower, the overall overhead is higher in applications that incur a high page relocation frequency. When relocation overhead falls on the execution's critical path — as in *cholesky* and *radix* — R-NUMA's performance relative to CC-NUMA+MigRep decreases. *Radix* incurs replacements in the page cache due to the cache's limited capacity and *radix*'s large primary working set of pages and therefore exhibits a large performance degradation due to high page relocation frequency.

In *barnes* and *fmm*, neither systems are sensitive to page operation overhead due to the low frequency of page operations. Page operations are not on *raytrace*'s critical path of execution and therefore do not affect performance much. In *lu*, page replication is susceptible to high overhead due to both replication and subsequent write faults to the replicated pages.

6.3 Sensitivity to Network Latency

Unlike DSMs with highly-customized memory systems like SGI Origin2000 [10], DSM clusters such as Sequent NUMAQ [12] often have a relatively large ratio of remote-to-local miss time. Techniques to reduce remote memory accesses can be more effective in DSMs with longer remote miss latencies. In this section, we evaluate the effect of longer remote miss latencies on CC-NUMA+MigRep's and R-NUMA's performance by varying the network latency.

Figure 7 illustrates the performance the systems for remote-to-local memory access latency ratio of 16, i.e., four times larger than our base system. The numbers are normalized with respect to perfect CC-NUMA. Not surprisingly, CC-NUMA's performance is highly sensitive to a larger network latency due to the large number of capacity/conflict misses. CC-NUMA's execution time on average increases to 126% (from 60% in the base system in Section 6.1) over perfect CC-NUMA. Application execution times on CC-NUMA+MigRep are less sensitive and on average increase to 72% over perfect CC-NUMA (as compared to 41% in the base system). R-NUMA incurs the least number of misses and exhibits an average execution time that is slightly over 25% as compared to perfect CC-NUMA (from 20% in the base system).

In *barnes*, *cholesky*, *lu*, and *raytrace*, CC-NUMA+MigRep significantly reduces the number of remote misses on the execution's critical path resulting in a commensurate reduction in communication time. R-NUMA virtually eliminates the remote misses in all these applications except for *cholesky* and performs best. In *cholesky*, R-NUMA simply relocates pages that are subsequently not used and therefore incurs the longer relocation overhead with the longer network latency. CC-NUMA+MigRep is least effective in *ocean*, *radix*, and *fmm*, and therefore the large number of misses directly increases the execution time with an increase in network latency in these applications. R-NUMA's performance shows slight sensitivity to a longer network latency in *ocean* and *radix* due to the high page relocation frequency and the resulting block fetch/refetches in these applications.

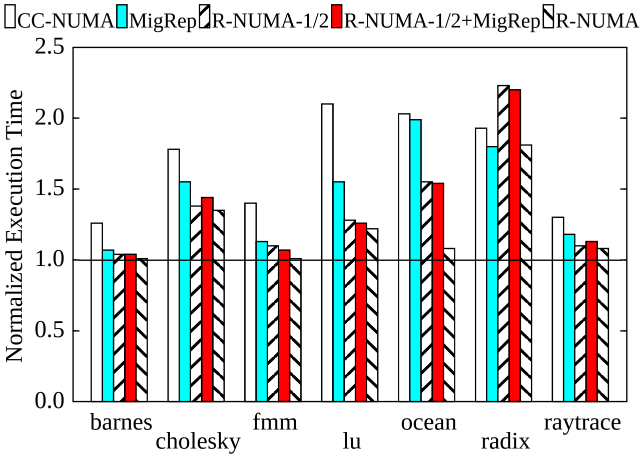

FIGURE 8. Performance achieved by R-NUMA+MigRep.

6.4 R-NUMA+MigRep

The results in Section 6.1 indicate that R-NUMA given an infinite page cache can eliminate all capacity/conflict traffic. R-NUMA's DSM hardware overhead — i.e., translation table, fine-grain tags, and counters — and cost, however, is a function of the page cache size. Current implementations of R-NUMA (e.g., Sun WildFire [6]), limit the hardware support to a small fraction of memory and provide only a "cache" of miss counters as opposed to per-page counters for all of memory.

In this section, we evaluate whether page migration/replication can be used to reduce the required page cache size and thereby design cost in R-NUMA. We present preliminary results on a system that integrates page migration/replication with R-NUMA, called R-NUMA+MigRep. The key to the integration is to design page migration/replication policies that can co-exist with R-NUMA's page relocation. The fundamental problem with the integration is that early relocation in R-NUMA prevents accurate page migration/replication because it reduces the capacity/conflict traffic and thereby impacts the CC-NUMA+MigRep's miss counters. Our system attempts to mitigates this problem by allowing page migration/replication to be invoked on every page for an initial preset time interval and delaying page relocation in R-NUMA. In this study, we allow R-NUMA relocation only after the first 32000 misses to a page.

Figure 8 depicts R-NUMA-MigRep's performance. The figure compares the performance of our base R-NUMA (with a 2.4 Mbytes page cache) with an R-NUMA-1/2 with half the page cache size, namely 1.2 Mbytes. The figure also plots R-NUMA-1/2+MigRep, adding page migration/replication to R-NUMA-1/2. All numbers are normalized against a perfect CC-NUMA. The figure indicates that R-NUMA-1/2's performance is not sensitive to MigRep. In *barnes*, *fmm*, and *raytrace*, R-NUMA-1/2 performs quite well and can not benefit much from further improvement. In *raytrace*, the extra delay in invoking page relocation slightly increases execution time for R-NUMA-1/2+MigRep as compared to R-NUMA-1/2. In *cholesky* and *lu*, page migration/replication happens throughout the application and page relocation in R-NUMA impacts the miss counters preventing page migration/replication from getting invoked in R-NUMA-1/2+MigRep. *Ocean* and *radix* do not benefit from either page migration or replication and therefore R-NUMA-1/2+MigRep performs as well as R-NUMA-1/2. These results indicate that integrating page relocation with page migration/replication requires more sophisticated mechanisms and policies that can identify candidates for page migration/replication for pages that are relocated to page caches.

7 Related Works

Verghese et al., [18], studied kernel-based page migration/replication in the context of both tightly-coupled CC-NUMA systems with low remote-to-local access ratios (around 3~5) and loosely-coupled systems with high remote-to-local access ratios (around 10~20). They studied both single parallel programs and multiprogrammed workloads with sequential and parallel applications. They evaluated page migration/replication for machines with no hardware or software support for page migration/replication and measured high overheads of around 400 µs for such operations. They suggest optimizations for stock SMP operating systems to reduce the overhead.

Soundararajan et el., [17] evaluated the flexibility of Stanford FLASH in implementing various remote caching strategies and optimizations including kernel-based page migration/replication using a large software remote cache in main memory. Their results are fundamentally different from ours in many respects. First, we only evaluate all-hardware protocol implementations with minimal coherence and memory access overhead. Second, we use selective fine-grain memory caching in hardware using R-NUMA and evaluate the required memory resources. They implement a software remote cache in memory using FLASH's memory miss handlers. Finally, as in a previous study [18], they only evaluate machines with no hardware support for page migration/replication. We also evaluate aggressive implementations with hardware counters and page invalidation and data gathering hardware support.

Falsafi and Wood first proposed and evaluated R-NUMA [5]. Moga and Dubois [14] studied remote caching in detail and carefully evaluated the design space for building block caches and R-NUMA. Hagersten and Koster evaluated remote caching in the Sun WildFire [6] which implements a variation of R-NUMA. S3mp [16], ASCOMA [8], and PRISM [4], also implement a hybrid of CC-NUMA and S-COMA. S3mp only allows statically selecting the caching policy for every page. ASCOMA implements selective fine-grain caching as in R-NUMA but always allocates S-COMA pages first. PRISM implements a software policy to switch between S-COMA and CC-NUMA but not vice versa. None of the above systems evaluated selective fine-grain memory caching as in R-NUMA with aggressive page migration/replication support to reduce capacity/conflict traffic.

8 Conclusions

In this paper, we compared and contrasted two techniques to improve capacity/conflict miss traffic in a conventional CC-NUMA system. Page migration/replication optimizes read-write accesses to a page used by a single processor by migrating the page to that processor and replicates all read-shared pages in the sharers' local memories. R-NUMA optimizes read-write accesses to any page by allowing a processor to cache that page in its main memory. Page migration/replication requires less hardware complexity as compared to R-NUMA, but has limited applicability and incurs much higher overheads even with tuned hardware/software support.

We evaluated the effectiveness of page migration/replication and R-NUMA in reducing capacity/conflict traffic in DSM clusters by executing shared-memory applications on simulated systems. Our results indicated that: (1) both page migration/replication and R-

NUMA substantially improve the system performance over ‘‘first-touch’’ migration in many applications, (2) page migration/replication has limited opportunity and can not eliminate all the capacity/conflict misses even with fast hardware support and unlimited amount of memory, (3) R-NUMA always performs best given a page cache large enough to fit an application’s primary working set and subsumes page migration/replication, (4) R-NUMA benefits more from hardware support to accelerate page operations than page migration/replication, and (5) integrating page migration/replication into R-NUMA to help reduce the hardware cost requires sophisticated mechanisms and policies to select candidates for page migration/replication.

References

[1] Luiz Andre Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 3–14, June 1998.

[2] Tony M. Brewer. A highly scalable system using up to 128 PA-RISC processors. In *Digest of Papers, COMPCON’95*, pages 133–144, March 1995.

[3] R. Clark and K. Alnes. An SCI interconnect chipset and adapter. In *Symposium Record, Hot Interconnects IV*, August 1996.

[4] Kattamuri Ekanadham, Beng-Hong Lim, Pratap Pattanaik, and Marc Snir. Prism: An integrated architecture for scalable shared memory. In *Proceedings of the Fourth IEEE Symposium on High-Performance Computer Architecture*, February 1998.

[5] Babak Falsafi and David A. Wood. Reactive NUMA: A design for unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229–240, June 1997.

[6] Erik Hagersten and Michael Koster. WildFire: A scalable path for SMPs. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 172–181, February 1999.

[7] Erik Hagersten, Ashley Saulsbury, and Anders Landin. Simple COMA node implementations. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, January 1994.

[8] Chen-Chi Kuo, John Carter, Ravindra Kuramkote, and Mark Swanson. Ascoma: An adaptive hybrid shared memory architecture. In *Proceedings of the 1998 International Conference on Parallel Processing*, August 1998.

[9] Rick LaRowe and Carla Ellis. Experimental comparison of memory management policies for numa multiprcessors. *ACM Transactions on Computer Systems*, 9(4):319–363, November 1991.

[10] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, May 1997.

[11] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica Lam. The stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[12] Tom Lovett and Russel Clapp. STiNG: A CC-NUMA compute system for the commercial marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[13] Michael Marchetti, Leonidas Kontothanassis, Ricardo Bianchini, and Michael L. Scott. Using simple page placement policies to reduce the cost of cache fills in coherent shared-memory systems. In *Proceedings of the Nineth International Parallel Processing Symposium*, April 1995.

[14] Adrian Moga and Michel Dubois. The effectiveness of SRAM network caches in clustered DSMs. In *Proceedings of the Fourth IEEE Symposium on High-Performance Computer Architecture*, pages 103–112, February 1998.

[15] Shubhendu S. Mukherjee, Steven K. Reinhardt, Babak Falsafi, Mike Litzkow, Steve Huss-Lederman, Mark D. Hill, James R. Larus, and David A. Wood. Wisconsin Wind Tunnel II: A fast and portable parallel architecture simulator. *IEEE Concurrency*, 2000. To appear.

[16] A. Nowatzyk, M. Monger, M. Parkin, E. Kelly, M. Borwne, G. Aybay, and D. Lee. S3.mp: A multiprocessor in a matchbox. In *Proc. PASA*, 1993.

[17] Vijayaraghavan Soundararajan, Mark Heinrich, Ben Verghese, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. Flexible use of memory for replication/migration in cache-coherent DSM multiprocessors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 342–355, June 1998.

[18] Ben Verghese, Scott Devine, Anoop Gupta, and Mendel Rosenblum. Operating system support for improving data locality on cc-numa compute servers. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 279–289, October 1996.

[19] Wolf-Dietrich Weber, Stephen Gold, Pat Helland, Takeshi Shimizu Thomas Wicki, and Winfried Wilcke. The Mercury interconnect architecture: A cost-effective infrastructure for high-performance servers. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, May 1997.

[20] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, July 1995.

[21] Zheng Zhang and Josep Torrellas. Reducing remote conflict misses: Numa with remote cache versus coma. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, pages 272–281, February 1997.