Appears in the 13th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'07), December 2007, Melbourne Austrailia

PAI: A Lightweight Mechanism for Single-Node Memory Recovery in DSM Servers

Jangwoo Kim[*] *Jared C. Smolens*[*] *Babak Falsafi*[*†] *James C. Hoe*[*]
[*]*Computer Architecture Laboratory (CALCM), Carnegie Mellon University*
[†]*School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne*
http://www.ece.cmu.edu/~truss

Abstract

Several recent studies identify the memory system as the most frequent source of hardware failures in commercial servers. Techniques to protect the memory system from failures must continue to service memory requests, despite hardware failures. Furthermore, to support existing OS's, the physical address space must be retained following reconfiguration. Existing techniques either suffer from a high performance overhead or require pervasive hardware changes to support transparent recovery.

In this paper, we propose Physical Address Indirection (PAI), a lightweight, hardware-based mechanism for memory system failure recovery. PAI provides a simple hardware mapping to transparently reconstruct affected data in alternate locations, while maintaining high performance and avoiding physical address changes. With full-system simulation of commercial and scientific workloads on a 16-node distributed shared memory server, we show that prior techniques have an average degraded mode performance loss of 14% and 51% for commercial and scientific workloads, respectively. Using PAI's data-swap reconstruction, the same workloads have 1% and 32% average performance losses.

1. Introduction

Commercial server systems based on distributed shared memory (DSM) architectures must provide reliable, high-performance operation, even in the face of hardware failures. Several recent studies identify the memory system as the most frequent source of failures in servers [12,16,19]. In a DSM, the memory address space spans all nodes; therefore the failure of any single component—including DRAM chips, memory modules and memory controllers—or an entire memory node results in data loss that affects the entire system. Downtime on critical server systems incurs significant financial costs [13]; therefore memory system failure tolerance is a key design requirement.

Memory system protection in DSMs must achieve two system-level goals. First, to maintain reliability and availability, the system must continue running without data loss following component failure in a single memory node. Furthermore, system must maintain performance while demapping and replacing the failed component. So long as availability and performance are maintained, data reconstruction can proceed gradually in the background before repair or replacement [1]. Second, operating systems (OS) necessarily depend upon an immutable physical address space to hold critical data structures such as pinned, physically-addressed and "unrelocatable" pages [19]. Therefore, even if underlying memory organization changes unexpectedly, existing OS's require the physical address space to remain unchanged.

Prior approaches either fail to meet these two requirements or require system-wide architectural changes to transparently tolerate changes in the physical memory organization. Distributed parity [5,15] reconstructs the contents of failed memory components using mechanisms similar to the RAID-5 commonly used with disk arrays [14]. While it provides strong reconstruction properties, distributed parity incurs a significant performance penalty when operating in *degraded mode* where lost data is reconstructed on demand. "Memory page retirement" [19] leverages OS virtual memory support to unmap damaged physical pages, but cannot tolerate faults that affect unrelocatable pages [9,11]. Alternatively, the absolute location of physical addresses can be managed by a hypervisor, however this requires significant changes to the system architecture and hardware and firmware changes throughout the design to efficiently support the memory reorganization [1]. Furthermore, because single nodes can be repaired before additional failures occur [16], simple reconstruction mechanisms can provide the same reliability benefits as a heavyweight, but flexible hypervisor architecture.

In this paper, we present *Physical Address Indirection* (PAI), a lightweight hardware-based mechanism for transparent reconstruction of single-node memory failures in DSM servers. PAI transparently isolates the failed component from the rest of the system, reconstructs lost values in other memory locations and reconfigures the memory system while avoiding changes to the physical address space. To preserve system availability and minimize performance overheads, PAI reconstructs lost data off the critical path of memory requests using distributed parity. The hardware changes to implement PAI are constrained to the directory controller, avoiding invasive modifications to the complicated processor core and OS.

We make the following contributions:

• **Physical Address Indirection (PAI)**. We introduce the concept of PAI, a lightweight hardware-based mechanism for memory system reconstruction.

• **Quantitative performance evaluation.** We evaluate PAI in using a full-system simulation of a 16-node DSM server with commercial database and web server workloads, as well as parallel scientific workloads, to show that:

1. Degraded mode causes a substantial performance loss for all workloads.

2. Two reconstruction modes enabled by PAI that have negligible overheads for commercial (avg.< 2%) and moderate overheads for scientific workloads (avg.< 32%).

This paper is organized as follows. In Section 2, we provide a background discussion and reconstruction goals. Sections 3 and 4 describe the PAI mechanism during error-free and reconstruction modes. Section 5 presents two reconstruction modes. We evaluate PAI in Section 6 and conclude in Section 7.

2. Distributed parity protection

This section discusses the vulnerabilities of the memory system to hardware faults and presents our fault model and reconstruction goals, describes a baseline DSM server protected by distributed parity and summarizes conventional memory protection techniques.

2.1. Memory system vulnerability

Hard errors are permanent hardware component failures of a hardware device, ranging from individual transistors to entire DIMM packages, memory controllers and memory nodes. Unlike soft errors (e.g., radiation-induced single-event upsets) which affect individual bits, hardware failures can cause small-scale multi-bit to large-scale gigabit data loss. These failures necessitate reconstruction mechanisms capable of rebuilding a large proportion of, or even all, the data stored on a memory node.

As server workloads increase their memory requirements, the total system reliability increasingly depends upon the reliability of its memory system components. A recent study [16] shows that memory system faults comprise the single most common failure

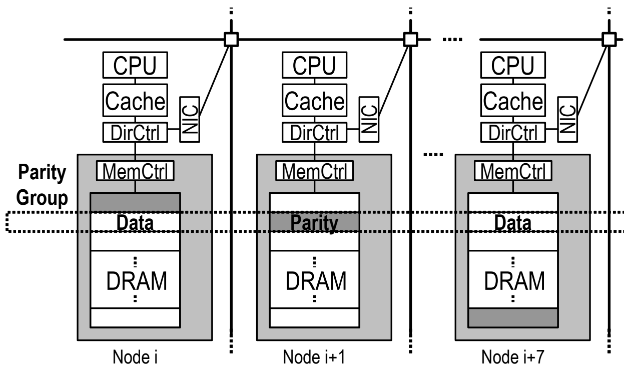

FIGURE 1. DSM server's memory system (shaded area) protected by (7:1) distributed parity.

mode during a nine-year study of a large installation of high-performance servers. Field failure data collected by Microsoft and Sun also show that memory contributes most frequently to system downtime for hardware failures on Windows and Sun servers [12,19].

2.2. Fault model

In this paper, we address reconstruction from hard failures originating in the memory controller and DRAM modules (shaded regions of Figure 1). We assume that memory includes SECDED ECC protection for soft error detection and correction and detection mechanisms can identify faults in the protected components (e.g., ECC detection on the memory arrays, self-checks and message time-outs), but that multi-bit errors are not recoverable with ECC protection alone. We assume that the remaining components in the system, including processors, caches, directory controller and NIC are protected by other, complementary mechanisms. For example, previously proposed processor redundancy [5] or checkpoint-based protection techniques [15,18] can be applied for non-memory component failures.

2.3. Distributed parity protection

The baseline system in this paper is the distributed shared memory (DSM) machine shown in Figure 1. Each node contains a processor with caches, a directory-based cache coherence controller, memory controller and associated DRAM channels, and a network interface to communicate with other nodes. Each memory node, consisting of the shaded memory controller, memory channels and DIMM modules, contributes to the system's large, single physical address (PA) space.

To support node-level reconstruction in DSM servers, recent studies propose distributing parity across nodes [5,15], as shown in Figure 1. These proposals arrange parity groups in a RAID level-5-like fashion [14], allowing parity group sizes that scale with the system size. Distributed parity has a scalable storage overhead which is amortized by distributing the parity group over multiple nodes. Distributed parity survives a single memory node failure per parity group by reconstructing the lost information in “degraded mode” with the bitwise XOR of the remaining data values and the parity value.

The system uses a distributed parity configuration as described by Prvulovic et al. in ReVive [15]. Parity is the bitwise XOR of all data stored in its parity group. Every data write updates the parity, on a remote node, with a new parity value obtained by bitwise-XOR of the old data, new data and old parity. Parity management is implemented by modifying the existing cache-coherence protocol and per-node directory controllers. In this paper, we assume parity is updated on a cache block granularity [5].

2.4. Recovery challenges with hardware faults

In degraded mode, distributed parity protection reconstructs data on each memory request. However, reconstruction compounds the memory latency problem for every value stored in the failed node. Each memory request incurs multiple additional requests to remote nodes to reconstruct the block, significantly reducing overall performance [5,15].

An ideal reconstruction mechanism instead rebuilds the lost data just once in an alternate location and redirects subsequent memory requests to the new location. However, relocation and message redirection based on changing the physical address is impossible when a failed component contains pinned pages—an unavoidable situation in large-scale failures. Although pinned pages consume a small fraction of the total PA space, field data collected from Sun servers using Solaris’s memory page retirement (MPR) show that in approximately 50% of failures, page retirement cannot be completed due to pinned pages in the damaged address range, leading to a system crash or data corruption [19]. Sun’s experience motivates a real need for memory systems that transparently relocate pinned pages. Alternatively, significant OS changes would be required to eliminate pinned pages [9,19].

2.5. Conventional memory protection

Hardware-based memory protection uses information redundancy to protect against failures. These mechanisms include ECC [17] and Chipkill [4], which protect against small numbers of failures in bits and single chips, respectively. However, these techniques neither protect against larger-scale component failure nor permit memory replacement without data loss. Board-level techniques such as memory mirroring, parity-based RAID and DIMM sparing [2,8] provide protection across DIMM modules on a single memory node. However, they have a fixed cost overhead, ranging from 33% for 3:1 RAID to 100% for memory mirroring, which cannot be amortized over the size of the system. Distributed parity [5,15] provides scalable protection, including reconstruction of data following the failure of an entire memory node, by spreading parity across nodes in the system and can amortize the cost over several nodes. However, degraded mode operation with distributed parity incurs a significant performance loss.

Other approaches include Solaris’s MPR [19], which demaps virtual pages that have been affected by a failure. This technique cannot restore lost data and cannot recover from failures that affect unrelocatable pages. In contrast, hardware hypervisor-based systems can perform flexible remapping and migration of OS-visible physical addresses memory pages to new absolute memory locations at runtime [1]. However, this approach requires architectural changes in all levels of the memory system to support page migration and dedicated hardware and firmware to map and manage the memory locations. Furthermore, while the hypervisor can move memory pages arbitrarily, this flexibility exceeds what is needed to survive a single failure in the memory system.

3. Physical Address Indirection

In this section, we introduce Physical Address Indirection (PAI), a lightweight mechanism that enables high-performance, transparent memory reconstruction, using distributed parity, following memory failures. Figure 2 shows a DSM system of 2N nodes protected by N:1 distributed parity. Memory from N+1 nodes forms a *parity group* where N nodes store data information and the remaining node stores parity (as in RAID-5 [14], parity is the bitwise XOR of all data values).

PAI provides a mapping between a physical address and its actual location (e.g., which memory node and parity group holds the data). PAI adds an additional level of address indirection for pages in memory, below that of the physical addresses seen by the OS. PAI comprises several control registers (*PAI registers)* with a simple, fast address translation logic between processor’s directory controller, memory controller and network interface.

Parity is arranged into parity groups where nodes are divided into two *sets* of N nodes apiece and are numbered in order of increasing PA. Each node is further divided into N+1 vertical equally-sized *sub-groups* where N hold data and one holds parity. The parity for each group resides in a different set from the data. To locate a group’s parity, a cross-set register (*X-Set*)

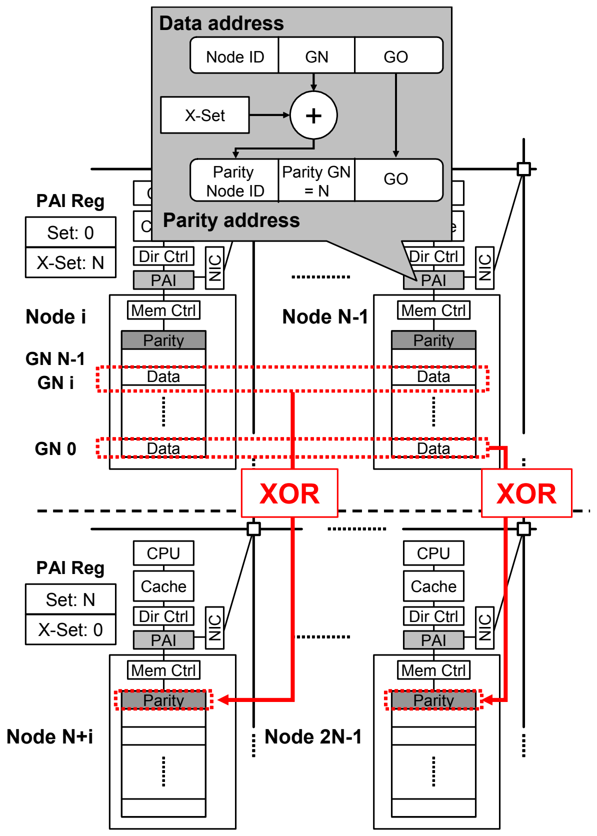

FIGURE 2. Distributed parity protection using PAI translations during error-free operation.

holds the number of the first node in the opposing set (e.g., in Figure 2, in Nodes 0 to N-1, X-set is N, while in nodes N to 2N-1, X-set is 0). This ensures that data and its parity never occupy the same node, and therefore, enough information always remains to reconstruct lost data or parity values after a single failure.

Figure 2 also shows the mapping from a cache block's PA to its corresponding parity. Each PA is divided into several logical fields that define the initial location of the cache block:

- Node ID
- Parity group number of the address (GN)
- Data address offset within the group (GO)

For each group, the parity node ID is calculated by summing the group number and the X-set register for that node. The parity sub-group is statically assigned to a sub-group on the node. To simplify presentation and without loss of generality, we choose the highest sub-group on each node to hold parity.

The parity sub-groups in PAI are arranged in a "horizontal" fashion across the set. In this arrangement, each sub-group occupies the same physical address offsets in each node. This contrasts with "diagonal" arrangements, such as Intel E8500 [8]. While diagonal parity offers better load balancing, it also makes recovery difficult with unrelocatable pages. In horizontal parity, each parity group is less likely to contain unrelocatable pages because these pages are typically located in a small range of fixed physical offsets in each node. By contrast, each diagonal parity group is likely to contain at least one unrelocatable page. Parity group arrangements that avoid unrelocatable pages are desirable for the data swap mode reconstruction presented in Section 5.

4. Reconstruction

This section describes PAI's reconstruction mechanism for surviving memory component failures.

4.1. Degraded mode reconstruction

When a memory component fails, the system enters *degraded mode*. In degraded mode, the distributed parity can reconstruct lost data from the remaining data and parity values. However, reconstruction imposes a significant performance overhead because every memory request on the failed node incurs multiple reconstruction requests. This situation persists until repair of the failed node. PAI also uses degraded mode reconstruction to restore lost data to alternate locations with a single mechanism. We explain how to locate reconstructed data in Section 5.

4.2. Memory system reconstruction

Following a memory component failure, a portion of the global physical address space disappears from the system. To recover to a high-performance state and restore parity protection, the faulty memory components must be mapped out and the affected pages must be moved to a fault-free memory node.

To reconstruct a failed node, PAI performs the operations shown in Figure 3. First, PAI coordinates on-line reconstruction of the lost data in alternate locations. While the reconstruction is performed in the background, PAI serves memory requests through degraded mode reconstruction or from new locations, depending on the reconstruction progress. Finally, when the reconstruction process completes, PAI resumes high-performance operation mode by redirecting subsequent requests to the reconstructed data. The failed memory is isolated from the rest of system.

Containment and reconstruction. PAI initiates on-line reconstruction of lost data according to the system reconstruction policy. PAI distinguishes already reconstructed regions from those pending reconstruction by maintaining a *Reconstruction Register* that contains the current reconstruction physical address in the directory controller of the failed memory node. Reconstruction is performed in the background, starting from the lowest physical address on the failed node and con-

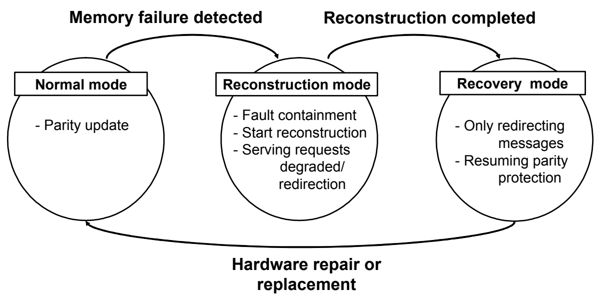

FIGURE 3. Steps in reconstruction and recovery of data from a failed memory node.

taining to the highest affected address. Following reconstruction, the failed component can be repaired.

Servicing requests during reconstruction. While reconstruction is performed in the background, PAI continues servicing memory requests. Requests are first sent to the respective home directory controller. PAI compares the requested address with the contents of the Reconstruction Register to determine the proper method for accessing the data. If the address is already reconstructed, the directory controller forwards the address to the PAI translation and completes the memory request, otherwise the directory controller issues a degraded mode request.

Leaving reconstruction mode. Once the reconstruction process has rebuilt all affected data for the failed node, the system enters recovery mode. In this mode, mode-specific PAI registers are updated with the failed node ID and recovery policy. Unlike reconstruction, where all memory requests are first sent to the failed home node, PAI minimizes the memory request latency by immediately translating the PAI address. Thus, memory requests go directly to the reconstructed node and the effective memory latency is the same as other remote requests.

5. Memory Recovery Modes

In this section, we present two recovery modes enabled by PAI: spare memory and data swap modes.

5.1. Spare Memory Mode

Spare memory mode utilizes a spare node with equal memory as the other nodes. This mode restores all data using parity reconstruction from the failed memory node onto the spare [17]. Using PAI, the system sends subsequent memory requests to the spare.

During error-free operation, PAs do not resolve to the spare memory node's ID. Upon entry into reconstruction, the data and parity values are reconstructed on the spare node by changing PAI's translation. PAI replaces the node ID of the failed node (Fail ID register) in the original PA with the spare memory node's ID. This remaps all PAs from the failed node without

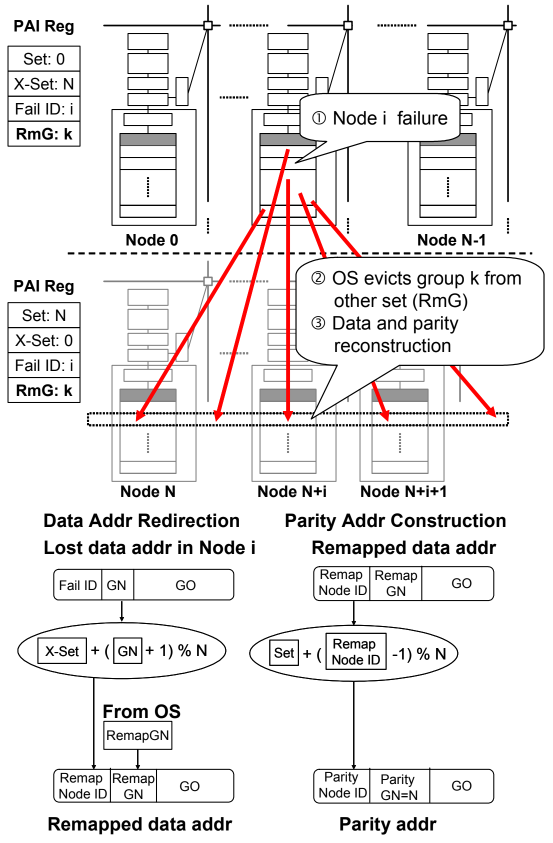

FIGURE 4. Data swap mode recovery rebuilds failed node i's data in an evicted group. The reconstruction process and updated PAI mappings are shown.

involving the OS and user programs or changing the amount of addressable memory in the system.

Spare memory nodes increase the cost of the system, however the overhead can be amortized over the size of the system. For typical DSM machines with four to sixteen memory nodes [3,10], the spare memory node overhead is lower than existing on-board redundancy mechanisms such as memory mirroring, RAID and DIMM sparing [2,8].

5.2. Data Swap Mode

Data swap mode maintains both redundancy and performance by reducing the available system memory, while incurring no additional hardware costs. Data swap mode leverages the OS's virtual memory support to free one parity group by swapping pages to disk or migrating the data to other free pages. Data swap mode reconstructs data from the failed node into the freed group, as shown in Figure 4.

Data swap mode must free an entire parity group. An ideal choice is to evict the parity group that lost its parity—the parity stored on the failed node—because this group is otherwise without protection. However, the OS may not evict this parity group if it contains pinned pages. The OS can instead choose another parity group with no pinned pages. The OS can find a free parity group easily because the memory uses horizontal parity groups.

Once the selected parity group is freed, data swap mode reconstructs data to the freed parity group. PAI then maps transparently requests for the failed node to the replacement parity group. By maintaining the original parity sub-groups, PAI preserves parity protection for the reconstructed data. Figure 4 summarizes the mappings for message redirection and parity address construction of remapped address.

Data swap mode exposes a subtle addressing restriction during recovery: data sub-groups cannot reside on the same node as their parity. Data swap mode avoids this problem by horizontally shifting the location of relocated data by one node, ensuring that data and parity are always separated.

Data swap mode retains the error correction capabilities of the baseline system, but reduces the available memory by one parity group. Data swap mode can also protect multiple node failures in one set with multiple PAI registers and further reducing memory. This is equivalent to the protection afforded by multiple spare memory nodes, without added hardware.

6. Evaluation

In this section, we evaluate the performance of PAI recovery modes.

6.1. Methodology

We evaluate the performance of PAI in FLEXUS, a full-system timing simulator [7]. We simulate a 16-node DSM running Solaris 8. Each node contains detailed processor core, microcoded directory controller, memory controller, and DRAM models. Nodes communicate using a directory-based NACK-free, 3-hop cache-coherence protocol over an interconnect based on the HP GS1280 [3]. We list the relevant parameters in Table 1.

We evaluate six commercial and two scientific workloads. We run both IBM DB2 v8 ESE and Oracle 10g Enterprise Database Server with an on-line transaction processing (OLTP) workload modeled after TPC-C. We evaluate a decision support system (DSS) workload modeled after TPC-H on DB2, using two representative queries: scan-bound query 1 and join-bound query 2. We evaluate web server performance using SpecWeb99 running on both Apache HTTP Server v2.0 and Zeus Web Server v4.3. We also evaluate two scientific workloads: *em3d* exhibits producer-consumer sharing between pairs of processors and *ocean* exhibits bursts of memory transactions. Our workloads are carefully tuned to produce representative memory system behavior.

We allocate pages to reduce remote misses and balance load: round-robin and first-touch for commercial and scientific workloads, respectively.

We use a systematic paired-measurement sampling approach derived from SMARTS [20]. We collect a sample size targeting 5% confidence intervals with measurements of 50,000 cycles, using warmed microarchitectural structures. For commercial workloads, we measure user-mode instructions committed, which is proportional to overall system throughput [6]. In scientific workloads, we measure the user-mode instructions committed by the slowest node over time, which reflects the overall execution time for the workload. In data swap mode, we assume the OS locates enough free virtual pages to absorb the decreased memory space, thus swapping to disk is unnecessary.

6.2. Recovery mode performance

In this section, we evaluate the performance of PAI's recovery modes after a hard failure in a single memory node. For data swap and spare memory modes, all data reconstruction and page migrations have taken place and the system is now operating in steady-state execution with one memory node inactive. Execution during the reconstruction process is similar to degraded mode, however the situation only lasts until reconstruction has completed.

Figure 5 shows the performance of degraded, data swap, and spare memory modes, normalized to error-free execution.

TABLE 1. DSM server and workload configuration.

Processing Nodes	UltraSPARC III ISA, TSO consistency 4 GHz, 8-wide, 8-stage; out-of-order 256-entry ROB, LSQ and store buffer
L1 Caches	Split I/D, 64KB 2-way, 2-cycle latency 4 ports, 32 MSHRs
L2 Cache	Unified, 8MB 8-way, 25-cycle latency 1 port, 32 MSHRs
Main Memory	60 ns latency, 2 channels, 32 banks 16 entry read/write queues per channel
Parity Protection	Two 8:1 parity groups for 16 nodes
Protocol Engine	1 GHz controller, 64 contexts
Interconnect	4x4 2D torus, 25 ns latency per hop 128 GB/s peak bisection bandwidth
Commercial Workloads	
DSS-DB2	10GB data, 3GB buffer pool
OLTP-DB2	10GB data, 64 clients, 450 MB buffer pool
OLTP-Oracle	10GB data, 16 clients, 1.4 GB buffer pool
Web	16K connections, fastCGI
Scientific Workloads	
Em3d	3M nodes, degree 2, span 5, 15% remote
Ocean	1026^2 grid, 9600s rlx, 20K res., err.tol.1e-

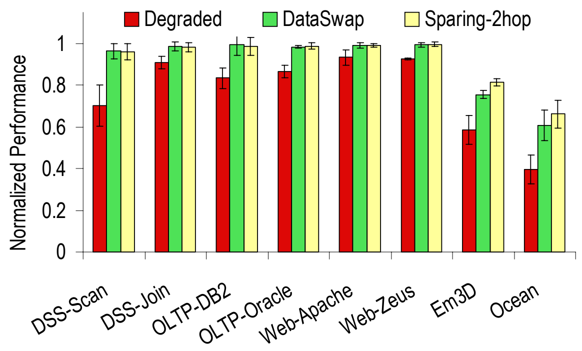

FIGURE 5. Recovery mode performance.

Degraded mode. Degraded mode suffers from a significant performance loss—averaging 14% for commercial workloads and 51% for scientific workloads. Every memory request to the failed node requires reconstruction of lost data and directory values. The failed node's performance is impacted because local memory must be reconstructed. The other nodes are degraded due to the increased remote latency of blocks owned by the failed node. Therefore, even though only a fraction of the address space is degraded, all nodes have a longer memory latency.

We now discuss this effect quantitatively. Table 2 summarizes the average frequency of L2 misses, writebacks, dirty misses (remote misses to modified data) and parity updates during error-free mode. Parity updates are divided into those from writebacks and dirty misses. The commercial workloads show a clear bias towards dirty sharing, while the scientific workloads exhibit frequent writeback requests. Therefore, during degraded mode the performance of commercial workloads will decrease because of remote memory accesses, while scientific workloads will be slow due to a high L2 miss rate.

This expectation is reflected in the execution time breakdown result during error-free and the recovery modes, in Figure 6. The execution time is divided into on-chip execution, time spent in spin-locks, stores, reads for private data and remote data. Compared to

TABLE 2. Workload memory characteristics per 1K cycles from baseline execution.

	L2 Misses	Write-backs	Dirty Misses	Parity Updates
DSS-Scan-DB2	0.78	0.01	0.28	0.29
DSS-Join-DB2	2.48	0.22	0.58	0.80
OLTP-DB2	3.95	0.30	0.54	0.84
OLTP-Oracle	5.18	0.57	0.44	1.01
Web-Apache	2.96	0.53	0.41	0.94
Web-Zeus	3.09	0.24	0.45	0.69
Em3d	7.94	1.85	0.02	1.87
Ocean	6.21	4.15	0.22	4.37

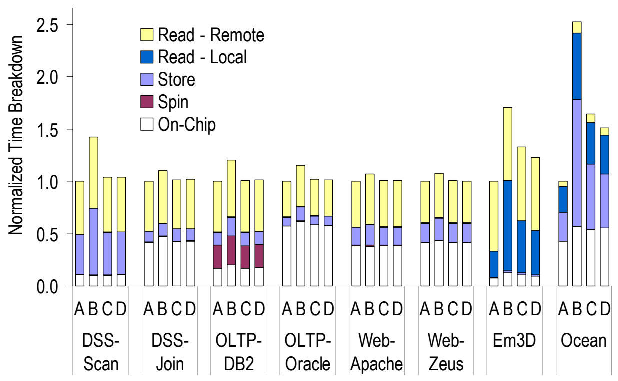

FIGURE 6. Execution time breakdown for (A) fault-free, (B) degraded, (C) data swap, and (D) sparing 2-hop modes, normalized to fault-free execution.

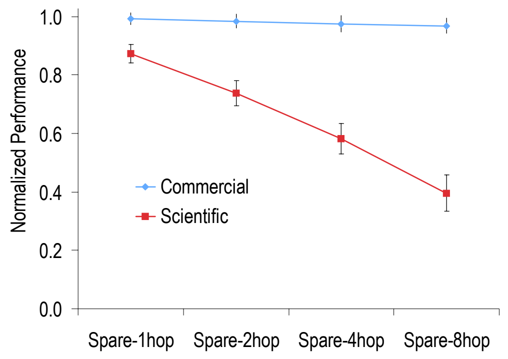

FIGURE 7. Performance sensitivity of spare node as average distance from the failed node changes.

error-free mode, commercial workloads spend additional time in off-chip remote read and write requests during the degraded mode. This reflects the increased latency for reconstructing shared data. *Em3d* spends additional time on local reads, due to the increased local memory read latency, while *ocean* spends additional time on stores, due to the increased local writeback latency.

Spare memory mode. We now study performance after reconstruction in spare memory mode. Figure 7 shows the performance when varying the distance between the failed and spare nodes. We vary from one network hop to eight hops to study effects beyond the maximum four hop distance in our interconnect.

The average performance impact at two hops is 2% for commercial workloads and 26% for scientific workloads. The primary overhead is increased memory access latency from the failed node's once-local data that is now on the spare. Other nodes do not experience significant performance loss because both directory and memory requests are already remote. Commercial workloads show little sensitivity because only a small fraction of the remote accesses are affected. However, performance for scientific workloads is severely degraded as the distance increases due to redirecting once-local memory requests to the remote spare.

Data swap mode. In data swap mode, PAI sends memory requests for the failed node directly to the reconstructed data locations. As with spare memory mode, only the local accesses to the lost memory observe increased latency from redirection.

The average distance between a failed node and the reclaimed space in other nodes is about 2.5 hops. Therefore, the performance of data swap mode is comparable to the performance of spare memory mode at 2.5 hops. The average performance impact is 1% and 32% for commercial and scientific workloads, respectively. Increased local memory access latency is the primary performance factor during degraded mode, therefore the performance of spare node at two hops and data swap mode are similar (as shown in Figure 6). For scientific workloads, data swap outperforms the spare memory mode at four hops and performs worse than spare memory mode at two hops. The slowdown comes from contention in nodes accepting additional memory requests, due to PAI redirection. By contrast, spare memory mode does not increase contention.

7. Conclusion

In this paper, we propose Physical Address Indirection (PAI), a lightweight, hardware-based mechanism for memory system failure recovery. PAI provides a simple hardware mapping to transparently reconstruct affected data in alternate locations, while maintaining high performance and avoiding physical address changes. With full-system, timing simulation of commercial and scientific workloads on a DSM server, we show that prior techniques have a substantial performance loss following hardware failure, while commercial workloads using PAI have a negligible performance loss.

Acknowledgements

We thank the anonymous reviewers and members of the SimFlex research group for their helpful comments on drafts of this paper. This work is supported by NSF CAREER award CCF-0347568, NSF award ACI-0325802, Sloan fellowships, the Center for Circuit and System Solutions (C2S2/FCRP), CyLab, and by grants and equipment from Intel Corporation.

References

[1] C. R. Conklin, C. J. Hollenback, C. Mayer, and A. Winter. Reducing planned outages for book hardware maintenance with concurrent book replacement. *IBM Journal of Research and Development*, 51(1/2), Jan-Mar 2007.

[2] C. C. Corporation. Compaq Advanced Memory Protection Technologies. *Compaq*, Jun 2002.

[3] Z. Cvetanovic. Performance analysis of the Alpha 21364-based HP GS1280 multiprocessor. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 218–229, June 2003.

[4] T. Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. *IBM Corp.*, 1997.

[5] B. Gold, J. Kim, J. Smolens, E. Chung, V. Liaskovitis, E. Nuvitadhi, B. Falsafi, J. Hoe, and A. Nowatzyk. TRUSS: A reliable, scalable server architecture. *IEEE Micro*, 25(6):51–59, Nov-Dec 2005.

[6] R. Hankins, T. Diep, M. Annavaram, B. Hirano, H. Eri, H. Nueckel, and J. P. Shen. Scaling and characterizing database workloads: Bridging the gap between research and practice. In *Proceedings of International Symposium on Microarchitecture*, Dec 2003.

[7] N. Hardavellas, S. Somogyi, T. Wenisch, R. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. Hoe, and A. Nowatzyk. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, 31(4):31–35, Apr 2004.

[8] Intel Corporation. Intel E8500 Chipset North Bridge (NB). *Intel reference manual*, Mar 2005.

[9] J. Jann, L. Browning, and R. Burugula. Dynamic reconfiguration: Basic building blocks for autonomic computing on ibm pseries servers. *IBM Systems Journal*, 42(1), Jan 2003.

[10] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron processor for multiprocessor servers. *IEEE Micro*, 23(2), March-April 2003.

[11] S. Microsystems. Predictive Self-Healing in the Solaris 10 Operating System - A Technical Introduction. *SUN Microsystems*, Jun 2004.

[12] B. Murphy. Automating software failure reporting. *ACM Queue*, 2(8), Nov 2004.

[13] D. Patterson. Recovery oriented computing: A new research agenda for a new century, Feb. 2002. Keynote Address, HPCA-8.

[14] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (raid). In *Special Interest Group on Management of Data*, pages 109–116, Jun 1988.

[15] M. Prvulovic, Z. Zhang, and J. Torrellas. Revive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proceedings of International Symposium on Computer Architecture*, May 2002.

[16] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. In *International Conference on Dependable Systems and Networks*, pages 249–258, Jun 2006.

[17] D. Sieworek and R. S. (Eds.). *Reliable Computer Systems: Design and Evaluation*. A K Peters, 3rd edition, 1998.

[18] D. Sorin, M. Martin, M. Hill, and D. Wood. Safetynet: Improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Proceedings of International Symposium on Computer Architecture*, May 2002.

[19] D. Tang, P. Carruthers, Z. Totari, and M. Shapiro. Assessment of the effect of memory page retirement on system ras against hardware faults. In *International Conference on Dependable Systems and Networks*, Jun 2006.

[20] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe. SMARTS: Accelerating microarchitecture simulation through rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.