Extending On-Chip Interconnects for Rack-Level Remote Resource Access

Yisong Chang*‡, Ke Zhang*‡, Sally A. McKee†, Lixin Zhang*, Mingyu Chen*‡, Liqiang Ren* and Zhiwei Xu*‡
*State Key Laboratory of Computer Architecture, ICT, CAS, Beijing, China
Email: {changyisong, zhangke, zhanglixin, cmy, renliqiang, zxu}@ict.ac.cn
‡University of Chinese Academy of Sciences, Beijing, China
†Chalmers University of Technology, Gothenburg, Sweden
Email: mckee@chalmers.se

***Abstract*—The need to perform data analytics on exploding data volumes coupled with the rapidly changing workloads in cloud computing places great pressure on data-center servers. To improve hardware resource utilization across servers within a rack, we propose Direct Extension of On-chip Interconnects (DEOI), a high-performance and efficient architecture for remote resource access among server nodes. DEOI extends an SoC server node's on-chip interconnect to access resources in adjacent nodes with no protocol changes, allowing remote memory and network resources to be used as if they were local. Our results on a four-node FPGA prototype show that the latency of user-level, cross-node, random reads to DEOI-connected remote memory is as low as 1.16μs, which beats current commercial technologies. We exploit DEOI remote access to improve performance of the Redis in-memory key-value framework by 47%. When using DEOI to access remote network resources, we observe an 8.4% average performance degradation and only a 2.52μs ping-pong latency disparity compared to using local assets. These results suggest that DEOI can be a promising mechanism for increasing both performance and efficiency in next-generation data-center servers.**

***Keywords*-data center; remote access; custom interconnect**

I. Introduction

The world is becoming increasingly connected, from the Internet of Things to personal devices, home/local networks, campus networks, and the internet. Hardware and software at all these levels generate huge volumes of data about what software we run when, where, and how, and about our online activities. The resulting deluge of data to analyze together with the need to simultaneously manage increasing numbers of applications [25] (representing diverse workloads with unpredictable characteristics [7]) stresses the capabilities of current data-center servers. In spite of these growing data-processing demands, more than 50-70% of memory resources are currently underutilized in Google data centers [31], [26], and Google's online-service data centers typically exhibit about 30% CPU utilization, on average [4]. Utilization rates are even lower in other industry data centers [20].

To address this mismatch, emerging *rack-scale computing* [10] servers relax boundaries between discrete machines so that individual server nodes can work together as a virtual, unified machine. Examples include the Intel® Rack Scale Architecture [16], Hewlett-Packard Moonshot [13], and U.C. Berkeley FireBox [3], in which compute, network, and storage resources can be efficiently modularized and remotely accessed within the same rack. These systems are built with commodity processor buses (e.g., QPI [17] or HT [1]), system buses (e.g., PCI Express or PCIe), network fabrics (e.g., Ethernet, InfiniBand (IB), or Converged Ethernet), and even custom interconnection infrastructures [28], [8], [37].

Most of these approaches target system scalability, energy/power efficiency (performance per watt), compute density, or user flexibility. Supporting cross-node, remote access to hardware resources is rarely efficient due to overheads from deep protocol stacks and cross-protocol conversions, non-transparent remote resources, and sophisticated software interfaces. Furthermore, it is difficult to remotely control high-performance I/O devices by directly accessing memory-mapped I/O registers and processing remote interrupt signals locally. Such inefficiencies cause remote resource access to be much slower than local access.

We propose Direct Extension of On-chip Interconnects (DEOI), an architecture and implementation for rack-level, inter-node remote resource access that allows remote assets to be transparently treated the same as local assets. DEOI eliminates the overheads of transaction conversions across protocol boundaries by directly extending the on-chip interconnect (OCI) of a local node's System-on-Chip (SoC) to adjacent intra-rack nodes. The hardware DEOI module is incorporated into the node's local coherence hierarchy to tightly integrate remote resources into the local node. The OS then allocates, maps, and manages remote resources in the same manner as local assets. This allows user-level applications to use remote hardware resources directly, without software modification.

Our current design allocates remote hardware resources exclusively to the requester node: donated remote memory

[1]This work is supported in part by the National Natural Science Foundation of China under grants number 61202057 and number 61521092, by the Strategic Priority Research Program of Chinese Academy of Sciences under grant number XDA06010401, and by the National Key Technologies Research and Development Program of China under grant number 2016YFB1000401. McKee is supported by a Visiting Scientist Award from the Chinese Academy of Sciences Presidential Fellowship Initiative.

[2]First authors Chang and Zhang have made equal contributions to this work and are listed alphabetically.

978-1-5090-5142-7/16/$31.00 ©2016 IEEE

regions cannot be shared simultaneously among other nodes inside the rack. This obviates the need for complex, inter-node memory consistency models. Our contributions are:

1) a simple DEOI architecture that receives remote access requests through the local, cache-coherent interconnect and directly delivers the responses via our unified, on-chip intra-node/inter-node interconnect (with no cross-protocol conversions);
2) architectural support for a load/store programming model that allows access to remote resources with no software modifications;
3) an ARM SoC-based FPGA evaluation platform that performs remote cache-line fills at roughly 4× the latency of local DRAM accesses and that suffers only an 8.4% performance degradation for using remote rather than local Ethernet interfaces; and
4) an application-level evaluation demonstrating that the capacity benefits of DEOI-connected, cost-effective remote memory improve prototype performance on an in-memory key-value framework by up to 47%.

II. Related Work

We briefly survey approaches to implementing extended memories and efficient inter-node communications. QPI [17], HT [1], and other NUMA-like processor buses can connect several CPU sockets among adjacent servers to allow processors to coherently access memory resources attached to other sockets, but implementing a unified, global physical address space requires that a single kernel image be deployed across participating nodes. This structure inhibits customization for heterogeneous components, limits scalability, and increases vulnerability to faults [34], [5], [28].

Many research efforts leverage remote memory to support large, in-memory applications. Hines et al.'s distributed Adaptive Network Memory Engine (Anemone) [14] allows pages to be swapped to the memories of remote nodes over Gigabit Ethernet, treating remote memory as another level in the hierarchy. This approach requires no software changes, but systems using standard socket I/O optimized for bulk transfers incur high network stack overheads at both the kernel and user levels. Lim et al.'s MICA [23] specifically accelerates in-memory key value (KV) stores, bypassing the kernel to implement a lighter-weight protocol stack and supporting parallel data access via a simple but lossy indexing scheme that maps keys to pointers.

Using RDMA over reliable connections instead of relying on protocol stacks like TCP to ensure reliable transmission can avoid many overhead costs. RDMA requests are sent directly to the NIC without involving the kernel and are serviced by the remote NIC without interrupting the CPU. In Dragojević et al.'s FaRM [9], applications use transactions to interact with KV objects in a shared address space, without regard for their location. FaRM uses lock-free reads over RDMA and ships functions (transactions) to the node storing the target object. This approach improves both latency and throughput relative to state-of-the-art distributed memory KV systems built over TCP/IP, but it requires modifying applications to use the FaRM API. Kalia et al. [19] borrow MICA's KV data structures to implement HERD, which improves on FaRM by avoiding one-sided RDMA reads and by using a mix of RDMA and (two-sided) messaging verbs to write client requests into remote server memory, which triggers the remote server to compute the reply. Minimizing network round trips increases throughput to the point that it saturates bandwidth on commodity RDMA hardware.

Anemone and MICA extend memory via Ethernet cards, and FaRM and HERD use RDMA cards. The network adapters in these extended-memory organizations manage all accesses to remote memory, making them invisible to the local core. Programming models such as Ethernet sockets or IB verbs must thus be used for cross-node transfers, which may necessitate software modifications. Since exploiting full bandwidth requires large transfers, fine-grained random accesses to remote memory are rarely well supported.

PCI Express (PCIe) is also increasingly used to connect small-scale, tightly coupled data-center racks that were once connected by more traditional HPC technologies [6]. Hou et al. [15]'s cost-effective data centers and Tu et al. [33]'s Marlin map the address spaces of PCIe-connected remote memory into the local system address space, allowing remote memory to be accessed via normal loads and stores. Differences between the system and PCIe domains necessitate that remote memory requests be redundantly (and expensively) converted between on-chip interconnect (OCI) signals and PCIe transaction layer packets (TLPs). Furthermore, since the kernel views PCIe-based remote memory as part of I/O space, remote accesses must use (inefficient) non-cacheable instructions.

Extending systems over commodity hardware lowers costs and may facilitate timely adoption, but adding dedicated hardware support delivers better performance and efficiency. Hanawa et al. [12] propose a custom network that uses PCIe to accelerate inter-node GPGPU communication, and Hamidouche et al. [11] build an IB-based framework for remote offload on Intel MIC clusters. Here remote devices are again invisible locally. In contrast, IBM's Coherent Accelerator Processor Interface (CAPI) [32] supports coherent memory access for accelerators, but it does not yet support multiple server nodes, and its design prevents access requests to remote resources from being forwarded to other servers.

Several proposed systems implement custom inter-node connections for servers intended to better support data-center and cloud workloads. Dong et al.'s Venice [8] prototypes multiple network interfaces on an FPGA platform that, like DEOI, implements remote memory access via both RDMA and normal load/store instructions. In contrast to DEOI, Venice accesses remote accelerators and NICs by exchanging inter-node data via message-based mailboxes

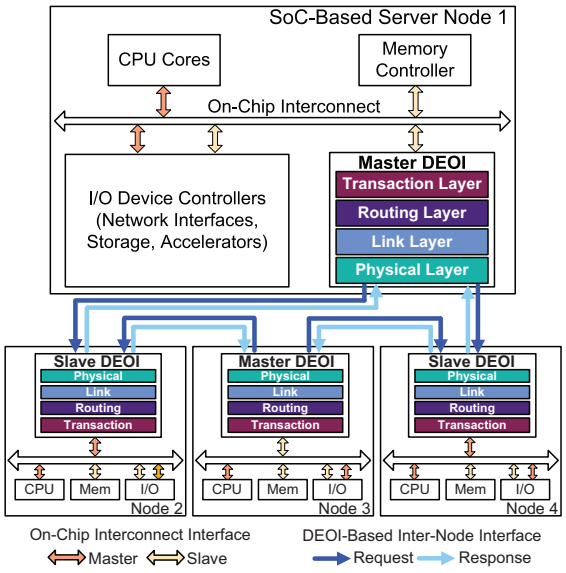


Figure 1: This shows the rack-level architecture for a system using DEOI to access remote memory and I/O resources.

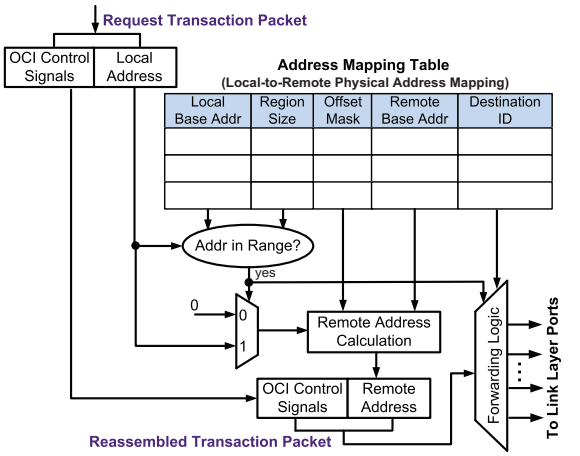


Figure 2: The master DEOI module's routing-layer Address Mapping Table (AMT) facilitates local-to-remote address translation and packet routing.

or QPairs: it does not support interrupt delivery or direct access to remote memory-mapped I/O registers. Scale-Out NUMA (soNUMA) [28] proposes an architecture, RDMA-inspired programming model, and communication protocol to support low-latency, distributed in-memory processing. Whereas performances and costs of Marlin [33], Venice [8], and DEOI have been evaluated via hardware prototypes, soNUMA has thus far been evaluated only in simulation.

Zhang et al. [37] address the capacity wall for in-memory applications in general by directly extending an on-chip AMBA AXI interconnect beyond the chip and transmitting data over high-speed serial lanes. This unifies the structures of the intra-node and inter-node connections, but the approach only supports a simple dual-node connection for remote memory access. Our DEOI specifically address the challenges of address mapping and multi-node connection for accessing both remote memory and remote I/O devices.

III. System Architecture

Fig. 1 shows how DEOI extends the server node's on-chip interconnect (OCI) logic to adjacent nodes within the rack. We integrate a four-layer DEOI module into the intra-node coherent interconnect as a custom, addressable device with a standard OCI interface. Master DEOI modules can directly connect to multiple slave modules and vice versa, although our current design limits connections to a single hop.

The transaction layer interacts with other local devices via the OCI. Tight integration allows this layer to process fine-grained, cache-line sized requests generated by CPU load/store instructions as well as coarse-grained data blocks whose transmission is initiated by other devices within the node. Signals on the OCI channels (address, burst length, data) are packaged directly into transaction packets to avoid overheads of repackaging the information with the additional headers and descriptors required by most inter-node protocols. Assembled packets are enqueued in per-channel asynchronous FIFOs to await routing.

The routing layer maps local addresses to their corresponding remote locations. This layer's header encodes the requesting node ID, and an Address Mapping Table (AMT), records the destination node ID. Packet forwarding logic uses the latter to map packets to the appropriate link-layer ports. The routing layer also monitors port congestion, throttling packets and applying back pressure to the transaction-layer FIFOs to implement flow control.

We implement the link and physical layers with standard IP components and use high-speed multi-gigabit SerDes technology to support maximum size per-channel transaction packets and to minimize pin-to-pin delays.

In order for local components (cores and I/O devices) to interact with remote resources, nodes must assign local addresses to borrowed assets. These local, physical addresses are *visible* in that they are what local components use to reference (otherwise invisible) remote resources. A portion of the physical memory space is set aside at design time for use as the node's locally visible address (LVA) window (2GB in our prototype). When a server node wants to borrow resources, it reserves an address block in its LVA window, allocates a slot in its AMT, and sends an out-of-band resource request to other intra-rack nodes. The donor returns the resource's base address and size to the requester, who updates the newly allocated AMT entry. The requester is the master DEOI node, and the donor is the slave.[3] Fig. 2 shows the structure of the DEOI master's AMT.

[3]We assume a monitor node manages remote resource allocation/donation, much as in the Venice system [8].

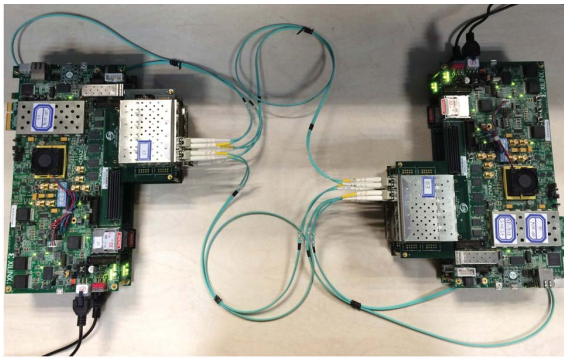


Figure 3: The four nodes in our ARM-based SoC prototype are connected by fiber and optical modules. We extend the FPGA's serial interface with a custom mezzanine card.

For outgoing requests, the routing layer looks in the AMT for the resource's physical address in the slave node and replaces the LVA with that address in the outgoing packet. Forwarding logic uses the destination ID (slave node ID) to place the packet on the appropriate link-layer port.

The master DEOI node uses normal memory and I/O instructions to access borrowed resources, and the OS manages access rights and allocation/deallocation as it would for local resources. Since the OS manages remote resources transparently, user-level applications remain agnostic to resource location. Loads and stores to remote memory are cached and kept coherent as if they were local. Unlike TCP sockets or IB verbs, this load/store programming model requires no software modifications, and access efficiency does not rely on remote data being accessed in large chunks.

Local cores can directly access memory-mapped I/O (MMIO) registers in remote devices. Unlike memory resources, I/O devices normally have a master on-chip interface for initiating DMA operations, and they raise interrupts to notify the cores on transfer completion. DEOI treats interrupt signals intended for remote nodes as packets arriving on an independent, virtual OCI channel. The master DEOI module can receive up to 16 independent interrupt signals from I/O devices within the local node. The interrupt ID is packaged with the interrupt signal to form a special type of OCI transaction packet. The master DEOI module arbitrates to transmit these interrupt packets at high priority when more than one I/O device asserts an interrupt signal. To maintain acceptable response times, DEOI drains the interrupt channel with highest priority when it detects a signal change in this channel. The DEOI master maintains an Interrupt Mapping Table (IMT) much like its Address Mapping Table. The IMT facilitates cross-node interrupt delivery by recording the relationships between interrupt IDs and destination node IDs.

Table I: Evaluation platform parameters

Hardware		
Component	**Remote Memory**	**Remote NIC**
CPU	Dual-Core ARM Cortex-A9@800MHz (32-bit)	
DRAM	1GB DDR3-1600 SODIMM	1GB DDR3-1066 component
1GigE	Zynq GEM MAC Marvell 88E1111 PHY chip	Xilinx AXI Ethernet MAC IP Xilinx 1000BASE-X PHY IP
DEOI	32-bit AXI4 I/F@200MHz, Xilinx Aurora link-layer protocol, x1 10Gbps high-speed serial lane	
Software		
OS	Linaro/Ubuntu 14.10 with Linux kernel 3.17	
Micro-benchmarks	Memory: user-level evaluation programs Network: iperf v3.0 [18], NetPIPE-3.7.2 [27]	
Applications	Redis v2.8.18 [29]	

IV. Evaluation

We study two representative resources for remote access — memory and Ethernet — and we implement an example DEOI system on a Xilinx® Zynq® ZC706 [36], an SoC platform with a dual-core ARM® Cortex™-A9 processor and programmable logic. Table I lists our system parameters.

The A9 cores support a full system software stack. We implement the DEOI hardware in programmable logic by using Xilinx standard cores for the memory controller and Ethernet interface and creating custom DEOI components. Fig. 3 shows our prototype with four SoC nodes connected by fiber and optical modules via the FPGA's high-speed serial interface, which we extend with a custom FPGA mezzanine card. Our prototype DEOI hardware conforms to the AMBA® AXI-4.0 protocol specification with the AXI Coherency Port (ACP) interface [2]. We use the Xilinx LogiCORE™ IP Aurora core and high-speed serial transceiver as the link and physical layers, respectively.

We use microbenchmarks to evaluate the raw performance of our DEOI prototype, and we run the Redis in-memory database framework to study system behavior for a real application. In particular, we stress the network by running microbenchmarks, iperf [18] (a tool for actively measuring achievable bandwidth on IP networks), and NetPIPE [27] (a protocol-independent performance tool) on a personal computer connected to the prototype via a standard Ethernet switch. We then run memtier_benchmark [30] (a high-throughput benchmarking tool for Redis and Memcached resources) as a key-value client on the PC. memtier_benchmark sends queries to a DEOI prototype node acting as the KV server.

A. Accessing Remote Memory

We use Linux Hotplug [24] to dynamically mount remote memory, and we invoke the `mmap()` system call to establish virtual-to-physical address mappings. The remote region can then be directly referenced from user-level applications. Our prototype allows us to probe each hardware module on the DEOI path, which lets us measure individual latencies

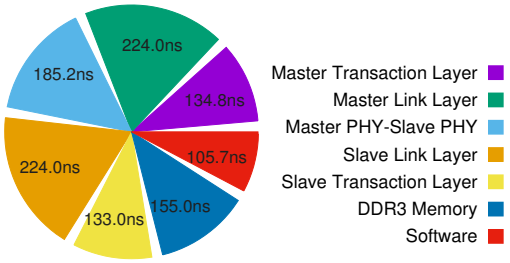


Figure 4: This breakdown of the round-trip time for a remote read to traverse the DEOI system stack shows that no individual component presents a bottleneck.

involved in remote resource access. Fig. 4 shows that the design of our DEOI system is balanced: latencies on the critical path are fairly equal, with no obvious bottlenecks. Total round-trip latency for remote reads is as low as 1.16μs.

Traversing the transaction layer in the master (slave) DEOI module takes 134.8ns (133.0ns), which accounts for only about 23% of the total RTT. The DEOI routing layer contributes no additional latency because the limited size of the AMT allows combinational logic to complete table lookup and packet reassembly in one cycle. Traversing the link and physical layers takes 224.0ns and 185.2ns, respectively, which accounts for 38.6% and 15.9% of the RTT. This indicates that the modular IP components in our evaluation platform introduce no additional pin-to-pin delay.

DEOI implements remote memory access at latencies within a small factor ($\sim$4$\times$) of accessing local memory.[4] In addition, our correspondence with a vendor specialist indicates that our FPGA-based DEOI prototype outperforms Mellanox's recent commercial ConnectX®-4 Virtual Protocol Interconnect® by 8.5% [22]. This bodes well for the performance potential of future DEOI systems based on even more powerful FPGAs or ASICs.

B. Accessing a Remote Network Interface

We next study the raw performance of accessing a remote Ethernet interface via DEOI. We configure iperf to run for 100 seconds over two parallel connections with an MTU size of 1500 bytes. Doing so generates stable inbound and outbound effective bandwidth and fully utilizes the dual-core CPU. Since large network I/O operations are segmented into packets smaller than the MTU, we evaluate ping-pong latency running NetPIPE with packets that are smaller than the default MTU size. We measure raw performance for local and remote Ethernet interfaces. We calculate the performance disparity d (as a percentage) for using local versus remote resources:

$$d = \frac{|p_{remote} - p_{local}|}{p_{local}} \times 100\% \quad (1)$$

[4]The local access latency is the sum of the DDR3 and software components in Fig. 4.

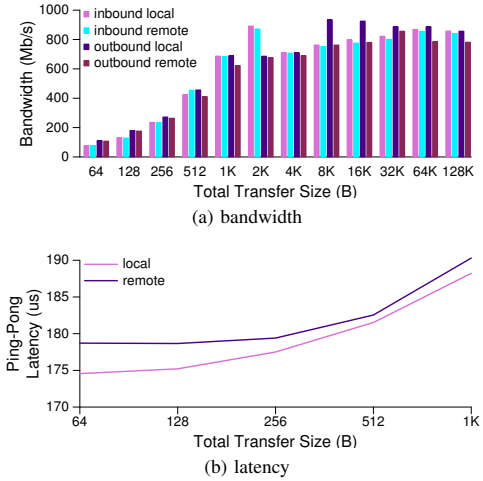


Figure 5: Effective bandwidths and latencies differ little for local versus remote Ethernet interfaces.

where p represents performance in terms of either bandwidth or latency. Obviously, smaller disparity values indicate more efficient DEOI access to remote I/O devices.

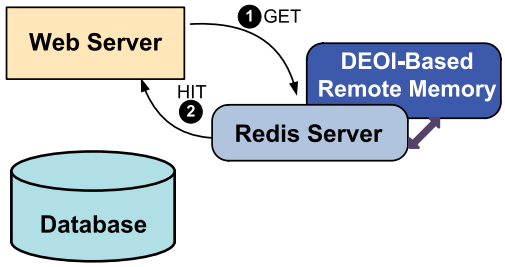
Fig. 5(a) compares the network bandwidth of local and remote Ethernet interfaces for different I/O data sizes and an MTU of 1500 bytes. Inbound and outbound effective bandwidths display average disparities of about 1.9% and 8.4%, respectively. Fig. 5(b) shows that using remote Ethernet incurs only 2.52μs additional ping-pong latency, on average, which represents but a small fraction of total latency.

C. Effects of Remote Memory on Application Performance

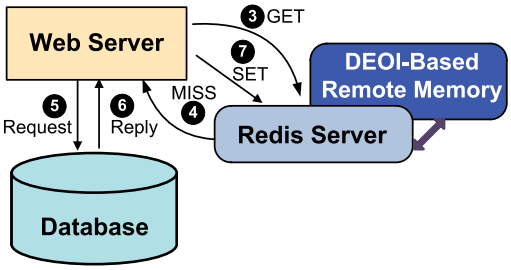
A typical in-memory key-value (KV) framework for interactive web services, Redis provides a fast, application-level caching tier between a web server and back-end database storage. We measure performance of interactive behaviors of Redis server between the Redis client (web server) and database storage. Fig. 6 illustrates a conventional Redis GET request sent from the web server over Ethernet. In Fig. 6(a), the request hits in the Redis server cache (❷). In Fig. 6(b), the request misses in the cache (❹), causing the web server to send a database query to the back end (❺) and await the reply (❻). A SET operation (❼) updates the Redis cache with the miss key and queried value.

We measure performance in terms of the average number of GET requests processed per second. We partition the Redis server's memory into two regions, a local *initial region* and a remote *additional region*. In our experiments, we set the capacity of the former to 1GB and that of the latter to 128, 256, 384, or 512MB. We first run a warm-up phase to exhaust the initial region, then we hotplug the additional region and monitor system behavior.

We simulate web server behavior with a modified version of memtier_benchmark [30], using the default query latency of 100μs on a Redis cache miss and the default 512B value



(a) Actions taken on a Redis cache hit



(b) Actions taken on a Redis cache miss

Figure 6: These diagrams illustrate the interactions between the Redis server, web server, and database storage (figures adapted from Lavasani et al. [21]).

field for our KV pairs. Fig. 7 shows per-second variations in performance. The greater the capacity of additional remote memory, the better the performance: Redis services up to 47.4% more requests per second with an extra 512MB of remote memory. KV pairs retrieved from the database back end get written to the Redis cache, filling remote memory early for the configurations with up to 384MB of remote memory (as evidenced by the drops in throughput at the left of the figure). Nonetheless, adding remote memory dramatically reduces the cost of evicting keys from the cache, letting the DEOI-extended systems sustain better throughput than the system that just uses local memory.

Redis performance is generally sensitive to the size of the key fields in the KV store and the latency for servicing a KV cache miss (i.e., the database query response time). We therefore study the performance impact of varying these parameters. Fig. 8 shows performances when using 512MB remote memory via DEOI relative to those when using 1GB of all-local memory. Using remote memory improves Redis performance for all three value sizes we test, even in the ideal case of instant query responses. The most dramatic improvements occur at the longest back-end query latency: even though processing larger value fields incurs more relative overhead, we still observe 1.14× better relative throughput at the maximum size/latency combination.

D. Hardware Cost

We use the Xilinx Vivado toolset [35] to carry out a full FPGA design flow. Our implementation results show that the

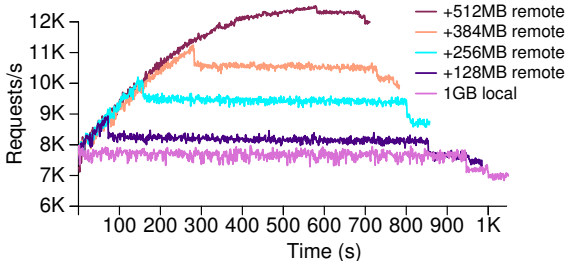


Figure 7: Redis throughput improves with increases in the amount of extra DEOI-connected remote memory used.

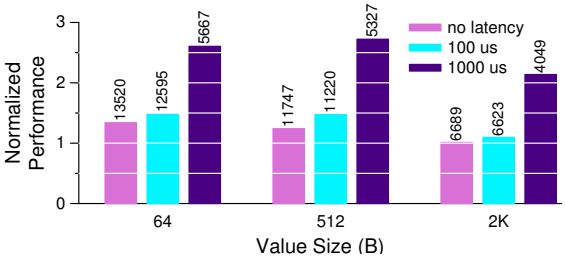


Figure 8: We study the impact of varying key field size and back-end database response time when borrowing 512MB of remote memory via DEOI. Results are normalized to those of the same Redis server using just 1GB of local memory, and numbers above the bars indicate average requests per second. Using DEOI-connected memory to increase the size of the in-memory KV store improves performance at all latencies (but results are most dramatic for long query response times) and lessens the impact of large key fields.

master DEOI module occupies only 1.2K slice LUTs and 2.4K slice registers, while the slave DEOI module occupies 1.3K and 2.5K, respectively. Each module consumes less than 1% of the logic resources in our evaluation platform's Xilinx XC7Z045 FPGA chip (which consists of 172K available LUTs and 344K available registers in total).

V. Limitations and Discussion

In our current design, remote memory is exclusively donated to a unique requester, whereas most distributed systems allow more flexible sharing. Flexibility usually comes at the cost of adopting user-level APIs for particular network fabrics, though. DEOI explicitly adopts a load/store programming model that requires no changes to application software. This model also facilitates cross-node sharing of pointer-based data structures (as opposed to requiring structures like trees and lists to be implemented with arrays). More flexible sharing in this model requires a more sophisticated, combined hardware/software solution.

We implement a quad-node DEOI prototype, but we evaluate only single-hop, point-to-point connections. This is due to the 32-bit address width of the current visible address window. Future platforms with 64-bit address buses will support more remote resources and a more sophisticated multi-

node interconnect, requiring either a centralized switching infrastructure or distributed on-chip routers together with QoS mechanisms based on specific topologies.

VI. Conclusion

We have proposed an efficient architecture and implementation for rack-level, cross-node resource access. We integrate memory and network resources into a local node's coherence hierarchy by extending the on-chip interconnect to directly interface with adjacent server nodes. Our approach avoids adding new headers and descriptors to on-chip interconncet packets and supports transparent access to remote resources with no changes to user-level software. Our results indicate that DEOI holds promise for future datacenter server architectures: the direct connections it supports can dramatically improve both throughput and efficiency.

References

[1] AMD, "HyperTransport 3.1 specification," http://www.hypertransport.org/docs/twgdocs/HTC20051222-0046-0035.pdf, [Online; accessed 25-March-2016].

[2] ARM, "AMBA AXI and ACE protocol specification," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html, 2013.

[3] K. Asanović, "FireBox: A hardware building block for 2020 warehouse-scale computers," in *Keynote Presentation of the USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2014.

[4] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, 2013.

[5] J. Chapin, M. Rosenblum, S. Devine, T. Lahiri, D. Teodosiu, and A. Gupta, "Hive: Fault containment for shared-memory multiprocessors," in *Proc. ACM Symposium on Operating System Principles (SOSP)*, Dec. 1995, pp. 12–25.

[6] L. Chisvin, "PCIe ready for datacenter role," EE Times Blog, http://www.eetimes.com/AUTHOR.ASP?SECTION_ID=36&DOC_ID=1319539, 2013, [Online; accessed 24-August-2016].

[7] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2013, pp. 77–88.

[8] J. Dong, R. Hou, M. Huang, T. Jiang, B. Zhao, S. A. McKee, H. Wang, X. Cui, and L. Zhang, "Venice: Exploring server architecture for effective resource sharing," in *Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Mar. 2016, pp. 507–518.

[9] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2014, pp. 401–414.

[10] B. Falsafi, T. Harris, D. Narayanan, and D. A. Patterson, "Rack-scale computing (Dagstuhl Seminar 15421)," *Dagstuhl Reports*, vol. 5, no. 10, pp. 35–49, 2016.

[11] K. Hamidouche, S. Potluri, H. Subramoni, K. C. Kandalla, and D. K. Panda, "MIC-RO: Enabling efficient remote offload on heterogeneous Many Integrated Core (MIC) clusters with InfiniBand," in *Proc. ACM International Conference on Supercomputing (ICS)*, Jun. 2013, pp. 399–408.

[12] T. Hanawa, Y. Kodama, T. Boku, and M. Sato, "Interconnection network for tightly coupled accelerators architecture," in *Proc. IEEE Symposium on High-Performance Interconnects (HOTI)*, Aug. 2013, pp. 79–82.

[13] Hewlett-Packard, "HP Moonshot system," https://www.hpe.com/us/en/servers/moonshot.html, [Online; accessed 25-May-2016].

[14] M. R. Hines, J. Wang, and K. Gopalan, "Distributed anemone: Transparent low-latency access to remote memory," in *Proc. International Conference on High Performance Computing (HiPC)*, Dec. 2006, pp. 509–521.

[15] R. Hou, T. Jiang, L. Zhang, P. Qi, J. Dong, H. Wang, X. Gu, and S. Zhang, "Cost effective data center servers," in *Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2013, pp. 179–187.

[16] Intel, "Intel Rack Scale Architecture: Faster service delivery and lower tco," http://www.intel.com/content/www/us/en/architecture-and-technology/intel-rack-scale-architecture.html, [Online; accessed 25-March-2016].

[17] ——, "An introduction to the Intel QuickPath Interconnect," http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html, [Online; accessed 25-March-2016].

[18] Iperf, http://software.es.net/iperf, [Online; accessed 25-March-2016].

[19] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proc. ACM Conference on SIGCOMM*, Aug. 2014, pp. 295–306.

[20] J. M. Kaplan, W. Forrest, and N. Kindler, "Revolutionizing data center energy efficiency," McKinsey & Company, Tech. Rep., 2008.

[21] M. Lavasani, H. Angepat, and D. Chiou, "An FPGA-based in-line accelerator for Memcached," *IEEE Computer Architecture Letters*, vol. 13, no. 2, pp. 57–60, 2014.

[22] Y. Li, "Personal communication (Mellanox Corp.)," Sep. 2015.

[23] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A holistic approach to fast in-memory key-value storage," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2014, pp. 429–444.

[24] Linux Memory Hotplug Support, http://lhms.sourceforge.net, [Online; accessed 25-May-2016].

[25] J. Mars, L. Tang, K. Skadron, M. L. Soffa, and R. Hundt, "Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proc. IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2011, pp. 248–259.

[26] A. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: Insights from Google compute cluster," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[27] NetPIPE, http://bitspjoule.org/netpipe, [Online; accessed 25-March-2016].

[28] S. Novaković, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-Out NUMA," in *Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2014, pp. 3–18.

[29] Redis, http://redis.io, [Online; accessed 25-March-2016].

[30] Redis Labs, "NoSQL Redis and Memcache traffic generation and benchmarking tool," https://github.com/RedisLabs/memtier_benchmark, [Online; accessed 25-March-2016].

[31] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of cloud at scale: Google trace analysis," in *Proc. ACM Symposium on Cloud Computing (SoCC)*, Oct. 2012, p. 7.

[32] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "CAPI: A coherent accelerator and development," *IBM Journal of Research and Development*, vol. 59, no. 1, 2015.

[33] C.-C. Tu, C.-T. Lee, and T.-C. Chiueh, "Marlin: A memory-based rack area network," in *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Oct. 2014, pp. 125–136.

[34] R. Unrau, O. Krieger, B. Gamsa, and M. Stumm, "Hierarchical clustering: A structure for scalable multiprocessor operating system design," *Springer Journal of Supercomputing*, vol. 9, no. 1, pp. 105–134, 1995.

[35] Xilinx, "Vivado design suite hlx editions," http://www.xilinx.com/products/design-tools/vivado.html, [Online; accessed 25-March-2016].

[36] ——, "Xilinx Zynq-7000 all programmable SoC ZC706 evaluation kit," http://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html, [Online; accessed 25-March-2016].

[37] K. Zhang, Y. Chang, L. Zhang, M. Chen, L. Yu, and Z. Xu, "sAXI: A high-efficient hardware inter-node link in ARM server for remote memory access," in *Proc. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 560–569.