Exploring the Disaggregated Memory Interface Design Space

Nathan Pemberton
nathanp@berkeley.edu
UC Berkeley

Abstract

Rapid increases in network performance, coupled with the dynamic nature of modern cloud workloads, has led to calls for memory disaggregation within the datacenter. In a memory-disaggregated machine, compute nodes contain only a limited amount of local memory, but are able to utilize free memory elsewhere in the machine over a network connection. This broad definition of memory disaggregation allows for a great diversity of techniques to actually provide that interface. In this paper, we present an overview of these techniques and lay out a few dimensions of the disaggregated memory interface design space. We conclude with a discussion of future research directions and challenges.

1. Introduction

Traditional data center designs aggregate all necessary resources (e.g., disk, memory, power supply, etc.) into many self contained server chassis. This design was motivated by the ability to leverage commodity PC components and networks [2]. Additionally, an aggregated design was desirable because in-chassis interconnects were significantly faster than networks. However, data center-side compute has grown into an important independent market, leading to specialized server platforms and networks (often called warehouse-scale computers (WSCs)). Furthermore, networking technology has seen a rapid increase in performance, with 40 Gbit/s Ethernet becoming commonplace, and 100+ Gbit/s networks readily available, narrowing the gap between off-package DRAM and remote memory [4]. Workloads have also changed; applications are fundamentally distributed (e.g., service-oriented architecture, map-reduce, etc.), use larger and rapidly changing datasets ("Big Data"), and demand latencies that can only be delivered by in-memory processing. Finally, a number of promising new memory technologies are becoming available. New non-volatile memory (NVM) devices are being introduced that promise low idle power, high density, and near-DRAM performance (e.g., fast NAND, phase-change, memristor) [9]. On the high-performance side, improvements in packaging technology have led to fast on-package DRAM (e.g., HBM) that offers hundreds of GB/s of bandwidth with capacities in the tens of GB [24].

These hardware and software trends have led to proposals from both academia [3, 23] and industry [16, 17, 18, 11] for a new style of WSC where resources are disaggregated. In a disaggregated WSC, resources like disk and memory become first-class citizens over a high-performance network. Compute nodes couple CPUs, network interfaces, and a small amount of high-speed memory into a self-contained system in package (SiP). This design allows data center operators to scale memory capacity, while allocating it more flexibly (avoiding stranded resources and complex resource allocation policies) [29]. However, the memory access latency will be higher than traditional off-package DRAM, and bandwidth may be limited or subjected to congestion. The small on-package memory allows us to mitigate some of this performance gap, but the question remains: how best to use it?

In this paper we try to bring structure to this question by presenting several dimensions of disaggregated memory interface design, and proposing a common framework for exploring this space. We describe these dimensions and list a number of additional desiderata in section 2. We then map a number of proposed interfaces into this taxonomy in section 3. Finally, in section 4, we explore a number of exciting future research directions and strategies.

2. Dimensions of Memory Interface Design

We begin by exploring two key dimensions that allow us to compare a wide array of techniques. Figure 1 places some common techniques in this space.

2.1. Explicit ↔ Implicit

The `Explicit\Implicit` dimension describes the level at which users must reason about local and remote memory resources. A fully `implicit` interface would require no applica-

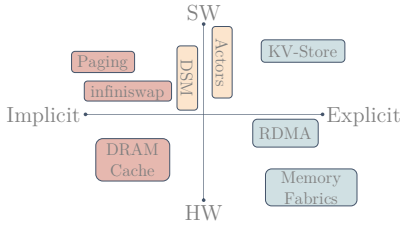

Figure 1: Two major dimensions of memory interface design. We place a number of common techniques in this design space.

In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported by gifts from Alibaba, Amazon Web Services, Ant Financial, Arm, CapitalOne, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

tion changes (relative to a single-node aggregated implementation) in order to operate *correctly* (performant code may still require modification). In contrast, fully `explicit` interfaces may require significant re-writing of applications in order to achieve both correctness and performance.

Implicit interfaces typically resemble caches. The system is automatically moving data from remote to local memory on-demand (perhaps with prefetching), and evicting old data to remote memory (either synchronously or asynchronously). We can borrow terms from the computer architecture community to further refine our understanding of a particular system. For example, caches can be write-back or write-through, inclusive or exclusive, coherent or incoherent. This approach also carries the well-studied limitations of caches such as requirements on temporal or spatial locality, maximum working set size, and metadata size vs block size. One effect of this similarity is that high-performance applications are already written to take advantage of caches, and many of the same optimizations will translate directly to the disaggregated memory setting. In general, users should find the `implicit` interface to be familiar, if not always intuitive.

Explicit interfaces more closely resemble scratchpads. These systems will require explicit movement of data from remote to local memory and vice versa. One implication of this strategy is that fully explicit systems require users to track their memory usage to avoid overflowing their local allocation (size miscalculations are a *correctness* issue). Scratchpads allow maximum flexibility for applications, allowing complex application-specific prefetching patterns and eviction policies. These interfaces also can be more efficient by avoiding excessive cache metadata or stranded resources (e.g. conflict misses). Distributed systems and data-intensive applications (e.g. an RDBMS) are likely to support `explicit` interfaces well as these systems already must account for limited local memory. Despite this, management of local memory remains a significant challenge for these applications.

Finally, it is important to note that this is not a strict dichotomy, but rather a spectrum of designs. Traditional hardware caches are fully `implicit`, but OS-managed paging allows some input from applications (e.g. mmap, pinning, etc.). One could also imagine a system that is `explicit` by default (e.g. RDMA), but degrades gracefully to `implicit` approaches (e.g. paging) as memory pressure increases.

2.2. Hardware Assisted ↔ Software Managed

Orthogonal to the `Explicit/Implicit` dimension is the level of hardware assistance provided by the remote memory interfaces. A fully `hardware assisted` system would perform all performance critical operations in hardware. A fully `software managed` system would use only the barest possible hardware interfaces (e.g. 'dumb NIC') to manage memory.

Hardware Assisted systems imply significant adoption barriers and may become rapidly outdated as systems evolve. However, hardware support can provide orders of magnitude improvement over software alone, particularly for latency-oriented tasks. A more subtle concern with hardware support comes from Myer and Sutherland's concept of the "wheel of reincarnation" [35]. In short, hardware does not come for free. Smart NICs often contain CPUs and non-trivial memory. At some point, we may pack enough sophistication into hardware-accelerated systems to have effectively re-invented the CPU. This can complicate analyses of total cost of ownership.

Software Managed interfaces benefit from flexibility and ease of deployment, but can suffer from significant performance limitations. It's worth noting that in many cases, careful systems engineering can overcome some of these limitations without introducing new hardware [27, 20, 15]. However, these solutions may require fragile systems tuning and complex control flow. Hardware has the potential to introduce a much simpler interface by offloading work to dedicated compute elements. Amazon's Nitro accelerator, for example, accelerates complex network virtualization [28].

2.3. Other Dimensions

In addition to the two main design dimensions, there are several other critical design decisions that any disaggregated memory interface must specify.

2.3.1. Durable ↔ Ephemeral Because remote memory is in a separate failure domain from node-local memory, its lifetime can be managed independently from any particular process. Varying levels of durability can be provided, from fully ephemeral (similar to memcached [33]), to atomically durable (like ramcloud [40]). Intermediate points, such as Pocket, are also available [25].

2.3.2. Isolated ↔ Fate Shared Related to durability, independent failure domains allow a system designer to choose the extent to which a failure will propagate. A totally `isolated` memory system will remain available regardless of any compute node. This behavior may be desirable for checkpoint/restart or systems with transactional semantics. However, we may choose to crash dependent compute nodes when their memory fails and vice versa (i.e. `fate share`). Recent work has shown how to track dependencies dynamically and execute flexible fate sharing policies [6].

2.3.3. Private ↔ Shared Finally, disaggregated memory can be connected to multiple compute nodes simultaneously. A fully `shared` system would allow fine-grained access to any memory location, by any compute node, at any time. In contrast, a fully `private` interface would allow no sharing at all, typically implying some level of ephemerality. In addition to durability and fate-sharing, shared memory must specify the access granularity in both time and space. For example, does the sharing occur at the level of bytes, pages, or some higher-level object? Must sharing occur in phases (e.g. 'ownership') or can it occur simultaneously (perhaps with explicit synchronization mechanisms)?

2.4. Other Requirements

In addition to the preceding dimensions, any new disaggregated memory will need to address at least the following questions:

1. **Naming:** How do tasks address remote memory? In the case of `ephemeral`, `private` data, this may be simple. Systems that support `sharing` and/or `durability` may impose non-trivial requirements on naming.
2. **Coherency and Consistency:** If `sharing` is supported, data may exist in multiple locations simultaneously and require special handling.
3. **Security/Fairness:** Disaggregation increases the degree of sharing in a system. A new design must describe mechanisms to protect applications from both malicious and accidental interference.

You may recognize two of the requirements from the famous quote (attributed to Phil Karlton): "There are only two hard things in computer science: Naming things, and cache invalidation [coherency]". Indeed, these issues require non-trivial thought.

3. Exploring the Disaggregated Memory Design Space

We now explore this design space by describing a number of established and novel memory interfaces. These examples were chosen to highlight interesting aspects of the disaggregated memory interface design space, a full survey of techniques is beyond the scope of this paper.

3.1. NUMA (`Implicit`, `HW Assisted`)

Non-uniform memory access (NUMA) architectures partition memory resources across several compute nodes such that all memory has the same interface (loads and stores from CPUs), but some is faster than others (non-uniform). Some NUMA systems include hardware services to aid in page migration to mitigate this effect [26].

The `implicit` interface presented by NUMA systems is appealing because they appear to software as a single, large memory. Because they are `hardware assisted`, they can also offer memory access latencies on the order of 100s of nanoseconds. This performance and tight coupling, however, limit scalability. The largest NUMA systems can scale to hundreds of nodes and 10s of TB of memory [41], but typical systems support only a few TB and less than 10 nodes (due to poor scaling in cost and power). These scalability and flexibility limitations are common in `(implicit,HW)` systems.

3.2. RDMA (`Explicit`, `HW Assisted`)

Remote direct memory access (RDMA) systems are similar to NUMA in that memory resources are partitioned among several compute nodes (memory is always local to someone) [39, 38]. The difference is that while NUMA systems typically expose an `implicit`, cache-coherent load-store interface to both local and remote memory resources, RDMA uses an `explicit`, put/get interface to access remote memory. Typically, this service is provided through the network interface (`hardware assisted`). The `explicit` interface allows RDMA systems to scale beyond NUMA to thousands of nodes and petabytes of memory, but this comes at the cost of slower remote memory access performance and a more complex interface to applications [32].

Typical of `hardware assisted` systems, RDMA has historically been considered costly and was primarily deployed in supercomputing environments, but recent Ethernet-based implementations have made them increasingly accessible [39].

3.3. Memory Semantic Fabrics (∗, `HW Assisted`)

A new class of interface has been recently introduced; the memory semantic fabric. A memory semantic fabric abstracts memory into a simple load-store interface (rather than technology-specific protocols). These `hardware assisted` interfaces are tightly coupled with the CPU, often loading memory directly into local caches. Unlike traditional NUMA, this abstraction enables heterogeneous memory technologies in flexible topologies. From the OS perspective, memory thus becomes an `explicit` first-class citizen on a memory-optimized interconnect and can be exposed to users as such. However, the OS may choose to make the interface `implicit`, typically through paging techniques (§3.4). Such interfaces promise to allow for greater scalability and flexibility than NUMA, while providing a less complex interface than RDMA. There are several commercial and academic projects developing cache-coherent interconnects for integrating accelerators and memories within a rack [8, 44, 37, 29]. Of particular note is an industrial effort called Gen-Z that provides a more general interface that can connect memory, accelerators, and storage using memory-oriented operations [13].

3.4. Paging and Page Migration (`Implicit`, `SW Managed`)

While memory semantic fabrics and RDMA offer an `explicit` interface at their lowest-levels, `software managed` abstractions like page migration can be added to make the interface more `implicit` [21]. In general, `explicit` interfaces are more general than `implicit` interfaces and can be abstracted through system software like the OS or language runtime.

NUMA systems typically expose a virtual memory abstraction to applications. The OS is responsible for choosing which NUMA domain to allocate memory from. This can be a complex decision and much effort has gone into studying such allocation policies [31].

More generally, operating systems may dynamically move pages between local and remote memory (or storage) via paging. This `(implicit, software managed)` interface effectively treats local memory as a cache for disaggregated memory. This approach is taken by many recent projects

in disaggregated memory due to the generality and ease of adoption `implicit` interfaces provide [12, 42, 14, 30].

Note that paging-based techniques vary in their degree of `hardware assistance` and `explicit` user control; they are not strictly in one camp or the other. Of particular note is the interface described by LegoOS, in which the authors describe a `hardware assisted` cache, and demonstrate the concept using a `software managed` technique [42]. This demonstrates the orthogonality of these dimensions. There are also variations along several dimensions listed in section 2.3. For example, the Mojim project uses virtual memory to present an (`isolated, durable, private`) interface to NVM [48]. Aguilera et. al. present an mmap-style approach that adds some `explicit` properties to paging based techniques and uses filesystem semantics to handle naming [1].

3.5. Language-Based Approaches (∗, `SW Managed`)

Some programming languages present a `software managed` interface with the illusion of a unified address space through the use of distributed shared memory. A particular design point in this space uses a partitioned global address space (PGAS) to make it appear as if some variables are `shared` while others are `private` [7, 36]. PGAS is partially `explicit` because users must choose which variables can be shared remotely while the language runtime `implicitly` handles data movement and caching. Actor models present another partially `explicit` interface where objects are explicitly shared through channels, but the language handles much of the complex sharing semantics described in section 2.4.

These languages are typically implemented such that they can take advantage of `hardware assistance` when available, but fall back to software techniques when needed [5].

3.6. Data Stores (`Explicit, SW Managed`)

Many systems expose remote memory through higher-level software constructs like databases and key value stores. Memcached is a widely used system for memory object caching that operates entirely in software [33]. Other systems use RDMA `hardware assistance` to accelerate access to these stores [40, 10]. Because these systems are typically `shared`, `isolated`, and (sometimes) `durable`, they must address the issues in section 2.4. Anna is one such system that provides a range of consistency semantics [46]. Lower-level `(explicit, HW)` interfaces may enable new abstractions as proposed by Volos et. al. [43].

4. Discussion

If we re-consider figure 1, we observe that the extreme corners of the design space are well explored by established techniques like paging and RDMA. Much of the recent academic work has thus focused on hybrid designs that trade off between the dimensions (see figure 2). For example, Infiniswap moves the `software managed` paging approach down the `HW/SW` dimension by utilizing RDMA-capable network cards [14]. In general, we believe that approaches that fix one dimension, while exploring another represent a significant opportunity for progress. Indeed, previously impractical approaches may become practical with changes to an orthogonal dimension. Programming models like serverless [19] or data center operating systems [42, 47] can help move designs along the `Implicit/Explicit` dimension, while recent advances in data center simulation will allow us to explore novel hybrid `HW/SW` designs (e.g. accelerators for paging) [22, 34].

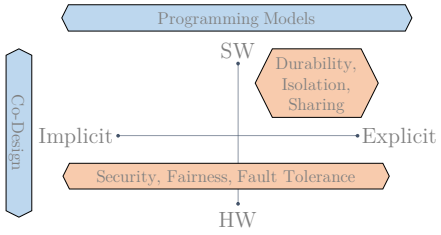

Figure 2: Research opportunities in the disaggregated memory design space. Sections 2.3 and 2.4 represent new research directions for the well-established corners of the design space. Programming models will allow for more hybrid designs across the Implicit/Explicit spectrum while HW/SW co-design can help bridge the HW/SW dimension. Finally, hardware-accelerated designs may enable new directions in security, fault tolerance, and QoS.

Although there is a significant body of work along the extreme corners of the design space, we believe that the dimensions laid out in sections 2.3 and 2.4 represent opportunities for future research in these regions. For example, recent work on key-value stores has explored the `durable`/`ephemeral` dimension [25], while others have continued the long tradition of research into memory consistency [46]. Finally, new hardware may enable progress on issues of security, fault tolerance [6], or fairness. For example, memory blades could authenticate memory requests using capabilities or access control lists [45].

Finally, the impact of other disaggregated resources must be considered. Disaggregated accelerators and storage may lack the sophisticated software stacks required to participate in `software managed` interfaces, while full `hardware assistance` may be prohibitively expensive on legacy nodes. Interfaces must provide a flexible interface to bridge this gap and provide a path to future upgrades.

5. Conclusion

In this paper we have identified two key dimensions in the design space of disaggregated memory interface design, `Implicit` ↔ `Explicit` and `Software Managed` ↔ `Hardware Assisted`. In addition, we identified several central design features including durability, isolation, and privacy. Finally, we argue that future research opportunities lie at the intersection of these dimensions, and that programming frameworks and HW/SW co-design are key to bridging that gap.

References

[1] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: a simple abstraction for remote memory. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 775–787, Boston, MA, 2018. USENIX Association.

[2] Thomas Anderson, David Culler, and David Patterson. The berkeley networks of workstations (now) project. In *Proceedings of the 40th IEEE Computer Society International Conference*, COMPCON '95, pages 322–, Washington, DC, USA, 1995. IEEE Computer Society.

[3] Krste Asanović. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *FAST 2014*, 2014.

[4] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. The end of slow networks: It's time for a redesign. *Proc. VLDB Endow.*, 9(7):528–539, Mar 2016.

[5] D. Bonachea and P. Hargrove. Gasnet specification, v1.8.1. Technical Report LBNL-2001064, Lawrence Berkeley National Laboratory, 2017.

[6] Amanda Carbonari and Ivan Beschasnikh. Tolerating faults in disaggregated datacenters. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 164–170, New York, NY, USA, 2017. ACM.

[7] William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. Introduction to upc and language specification. Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, 1999.

[8] Cache coherent interconnect for accelerators. https://www.ccixconsortium.com, 2017.

[9] An Chen. A review of emerging non-volatile memory (NVM) technologies and applications. *Solid-State Electronics*, 125:25–38, 2016.

[10] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. *No Compromises: Distributed Transactions with Consistency, Availability, and Performance*, page 54–70. ACM, 2015.

[11] Facebook. Disaggregated rack. http://www.opencompute.org/wp/wp-content/uploads/2013/01/OCP_Summit_IV_Disaggregation_Jason_Taylor.pdf, 2013.

[12] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network Requirements for Resource Disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 249–264, Savannah, GA, 2016. USENIX Association.

[13] Gen-Z Consortium. Gen-z overview. Technical report, Gen-Z Consortium, 2016.

[14] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, 2017. USENIX Association.

[15] Nadav Har'El, Abel Gordon, Alex Landau, Muli Ben-Yehuda, Avishay Traeger, and Razya Ladelsky. Efficient and scalable paravirtual i/o system. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 231–242, San Jose, CA, 2013. USENIX.

[16] HP Labs. The machine. https://www.labs.hpe.com/the-machine, 2017.

[17] Huawei. High throughput computing data center architecture - thinking of data center 3.0. www.huawei.com/ilink/en/download/HW_349607, 2014.

[18] Intel. Intel rack scale design. https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html, 2017.

[19] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A Berkeley view on serverless computing, 2019.

[20] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 185–201, Savannah, GA, 2016. USENIX Association.

[21] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. *HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter*, page 521–534. ACM, 2017.

[22] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Bora Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanovic. Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42, June 2018.

[23] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 690–695, March 2016.

[24] Joonyoung Kim and Younsu Kim. Hbm: Memory solution for bandwidth-hungry processors. In *2014 IEEE Hot Chips 26 Symposium (HCS)*, pages 1–24, Aug 2014.

[25] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 427–444, Carlsbad, CA, 2018. USENIX Association.

[26] James Laudon and Daniel Lenoski. *The SGI Origin: A ccNUMA Highly Scalable Server*, page 241–251. ISCA '97. ACM, 1997.

[27] Jialin Li, Naveen Kr. Sharma, Dan R. K. Ports, and Steven D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 9:1–9:14, New York, NY, USA, 2014. ACM.

[28] Anthony Liguori. C5 instances and the evolution of amazon ec2 virtualization. In *AWS re:INVENT*, 2017.

[29] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 267–278, New York, NY, USA, 2009. ACM.

[30] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F. Wenisch. *System-level implications of disaggregated memory*, page 1–12. IEEE, 2012.

[31] *What is Linux Memory Policy?*, 2017.

[32] *Deploying HPC Cluster with Mellanox InfiniBand Interconnect Solutions*, 2017.

[33] memcached: a distributed memory object caching system, 2019.

[34] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim. dist-gem5: Distributed simulation of computer clusters. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 153–162, April 2017.

[35] T.H. Myer and I.E. Sutherland. On the design of display processors. *Communications of the ACM*, 11(6), 1968.

[36] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, page 291–305, 2015.

[37] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. Scale-out NUMA. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 3–18, New York, NY, USA, 2014. ACM.

[38] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A remote direct memory access protocol specification. RFC 5040, RFC Editor, October 2007.

[39] RoCE in the Data Center, October 2014.

[40] Stephen M. Rumble, Ankita Kejriwal, and John Ousterhout. Log-structured memory for dram-based storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, pages 1–16, Santa Clara, CA, 2014. USENIX.

[41] Sgi uv 3000, uv 30: Big brains for no-limit computing. https://www.sgi.com/pdfs/4555.pdf, 2016.

[42] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, Carlsbad, CA, 2018. USENIX Association.

[43] Haris Volos, Kimberly Keeton, Yupu Zhang, Milind Chabbi, Se Kwon Lee, Mark Lillibridge, Yuvraj Patel, and Wei Zhang. Memory-oriented distributed computing at rack scale. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, pages 529–529, New York, NY, USA, 2018. ACM.

[44] Bruce Wile. Coherent accelerator processor interface (capi) for power8 systems. Technical report, IBM Systems and Technology Group, September 2014.

[45] Jonathan Woodruff, Robert N.M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. The cheri capability model: Revisiting risc in an age of risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, page 457–468. IEEE Press, 2014.

[46] C. Wu, J. Faleiro, Y. Lin, and J. Hellerstein. Anna: A kvs for any scale. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 401–412, April 2018.

[47] Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. The datacenter needs an operating system. In *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2011.

[48] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. *Mojim: A Reliable and Highly-Available Non-Volatile Memory System*, page 3–18. ACM, 2015.