CacheCloud: Towards Speed-of-light Datacenter Communication

Shelby Thomas
UC San Diego

Geoffrey M. Voelker
UC San Diego

George Porter
UC San Diego

Abstract

The network is continuing to advance unabated, with 100-Gb/s Ethernet already a commercial reality, and now 400-Gb/s in the standardization process. Within a single rack, inter-server latency will soon be in the range of 250ns, trending ever closer towards the fundamental propagation delay of light in a fiber. In this paper we argue that in this environment, a major performance bottleneck is DRAM latency, which has stagnated at 100ns per access. Consequently, data should be kept entirely in the CPU cache which has an order of magnitude lower latency and RAM should be considered a slower backing store. We describe the implications of designing a "speed of light" datacenter network stack that can keep up with ever increasing link speeds with the goal of keeping latency as close to the speed of light propagation time as possible.

1 Introduction

Large-scale operators and cloud providers have architected highly parallel storage systems, data analytics frameworks, cluster managers and schedulers, and user-facing applications. Yet processing an individual request inevitably still relies on some amount of serial execution, and as a result subsystem latency has become critical, in line with Amdahl's Law. Perhaps more important for cloud providers is processing as much data as possible per user request, which likewise requires low latency, referred to as Gustafson-Barsis's Law [10]. The need for low-latency networked applications is rooted both in ensuring good user experiences, and also in maximizing the amount of data processed per request.

The performance characteristics of system components evolve independently. Perhaps a decade ago, the network throughput of gigabit Ethernet was typically much lower than what could be supported by multi-core CPUs and their associated memory (e.g., up to 10 Gb/s). This led to the evolution of interrupt-driven network stacks (in the 90s and early 2000s). The end of Moore's law and Dennard scaling has meant that CPU speed has largely flattened, leading to more cores (but not higher per-core performance). With the introduction of 10- and 40-Gb/s Ethernet, the network has largely caught up, and is nearing parity with the performance of the CPU and memory. This has led to the adoption of user-space network stacks like DPDK [8] and Netmap [29], which bypass the kernel and typically rely on dedicated cores to handle packet processing.

The network is continuing to advance unabated, with 100-Gb/s Ethernet already a commercial reality, and now 400-Gb/s in the standardization process, paired with both research [28] and commercial [36] demonstrations. At these speeds, links are typically optical, due to cost and energy constraints. The result will be a datacenter network that can deliver traffic at the speed of light (in a fiber) at bandwidths that exceed those of the endhost. To the endhost, the network will appear to deliver data at near zero latency and with infinite bandwidth. In the restricted environment of cluster and datacenter networks, the network community will have finally achieved their end goal of hitting the fundamental physical limits of communication performance.

Where does this leave server, OS, and application design? For servers communicating between racks at typical distances of O(50m), the one-way latency will be in the range of 250ns, and for intra-rack communication, the latency could be an order of magnitude smaller. This one-way latency represents one "component" in our system that is fundamentally impossible to optimize or improve, at any cost. The challenge then is to design a server architecture, network stack, and application that can meet this fundamental lower bound set by the network.

We have two options. First, we can continue to design systems that rely on parallelism to hide (but not reduce) this latency. Systems today adopt this strategy, since they are equipped with many-core CPUs, multiple memory banks consisting of multiple parallel channels, and high-speed network interfaces. Under this strategy, parallelism increases throughput but does not decrease the latency of an individual request. Here a major bottleneck is RAM latency, which has stagnated at 100ns per access [19]. Because of this bottleneck, we propose a second option: reducing the latency of an individual request by eschewing RAM in part or entirely! The idea is to move data out of RAM and keep it in the CPU cache itself, which has an order of magnitude lower access latency than RAM. This both lowers the latency of individual requests and increases total throughput. Eliminating the "RAM latency bottleneck" results not only in lower application latency, but also has implications for fault tolerance, consensus, lock managers, atomic counters, and RPCs–most of which operate one or two orders of magnitude below the speed

	Length (in m)	Latency (in ns)
Most common	20	100
50th-percentile	40	200
90th-percentile	110	550
Maximum length	150	751
DRAM		100
L3 Cache		12

Table 1: **Lengths of fiber optic cables in modern datacenters [7] and associated propagation delay compared with the memory hierarchy.**

of light between the millisecond to microsecond regime.

In this paper we argue that keeping datacenter communication close to the speed of light propagation time necessitates the need for *CacheCloud*, a cluster-wide SRAM manager that treats DRAM as a slower backing store and SRAM as a first class citizen.

2 Motivation

Improving the latency of networked software is not a new goal. Indeed, for decades providers have worked to reduce subsystem latency. In this section, we motivate why fiber propagation delay is the right metric to use as a baseline in subsystem performance. We then describe why the recent developments in hardware, the operating system, and software stack motivate this new baseline at this point in time.

2.1 Why propagation delay as a baseline?

Singla et al. [32] outlined the case for why the speed of light serves as an aspirational latency goal in the wide area. Indeed in wide-area networks, propagation delay dominates communication latency, and so focusing on matching the end-to-end delivery time to the underlying propagation delay aligns well. But what about clusters and datacenter environments? We know that for large-scale providers, microseconds count when targeting improved latency [4]. How do these expressed goals compare to the speed of light in a fiber?

Table 1 reproduces some reported measurements of observed fiber-optic cable lengths in datacenters, using multimode fiber (the distribution of single mode fiber is similar). At 100 Gb/s, the maximum supported fiber lengths are 150m, the average length (and latency) observed was just over 50m (271 us), and the 90th percentile is a bit over 100m (550 us). The most frequently observed length was only 20m (100 us) [7]. While each datacenter will be unique, the takeaway from this published study is that for many server pairs, one-way latencies of 100-200 μs are very possible.

Packet Size	100G DRAM	100G L3	400G DRAM	400G L3
64	20	2	79	10
128	10	1	40	5
192	7	1	27	4
256	5	1	20	3
320	4	1	16	2

Table 2: **Number of physical cores needed to process packets at line rates for 100G and 400G links for a single DRAM access. Compared to a single L3 (SRAM) access, a network application with just 1 DRAM access require a 4–8x increase in physical core count which translates to a 2–10x increased server cost based on current Intel Xeon SKUs.**

2.2 Why speed of light latency *now*?

A number of recent advancements make this the right time to strive for ultra-low latency.

Datacenter networks consist not only of individual fibers, but rather a topology of switches interconnected into multi-hop end-to-end links. For this reason, the end-to-end latency is also a function of the number of hops (and associated queuing delay) in the topology. For folded-Clos "Fat-Tree" topologies, the end-to-end hop count is typically uniform, with perhaps 5 to 9 hops between servers in disjoint parts of the network [1]. Recent proposals for Expander graph based topologies have the potential for greatly reducing the average hop count observed by flows in the network [16, 33, 34]. The end result will be less inherent latency since traffic will transit fewer switches. Thus propagation delay will increase as a key source of end-to-end latency.

The evolution of hardware and network capabilities shows us that we are reaching the beginning of a new era that brings computer architecture and network closer than ever before. As links have gotten faster, the inter packet gap has shrunk but internal hardware performance such as DRAM and cache latency has stayed mostly constant. This is due to the tendency for hardware manufacturers to prefer scaling out rather than reducing latency [27].

For years, DRAM has been sufficient for our networking needs as the interpacket gap has stayed well above DRAM latency and at one point with 1G links, above disk latency. Figure 1, however, paints a different picture for the future. Only for MTU sized packets at 100G the interpacket gap is above DRAM latency, meaning that cache misses from the L3 cache can be tolerated as memory reads are faster than packet arrival. This changes when we look beyond 100G. 400G links represent the cusp of a new era with network latencies finally exceeding DRAM latency for **all packet sizes**, whether minimum-sized (64 bytes) or MTU (1500 bytes or higher).

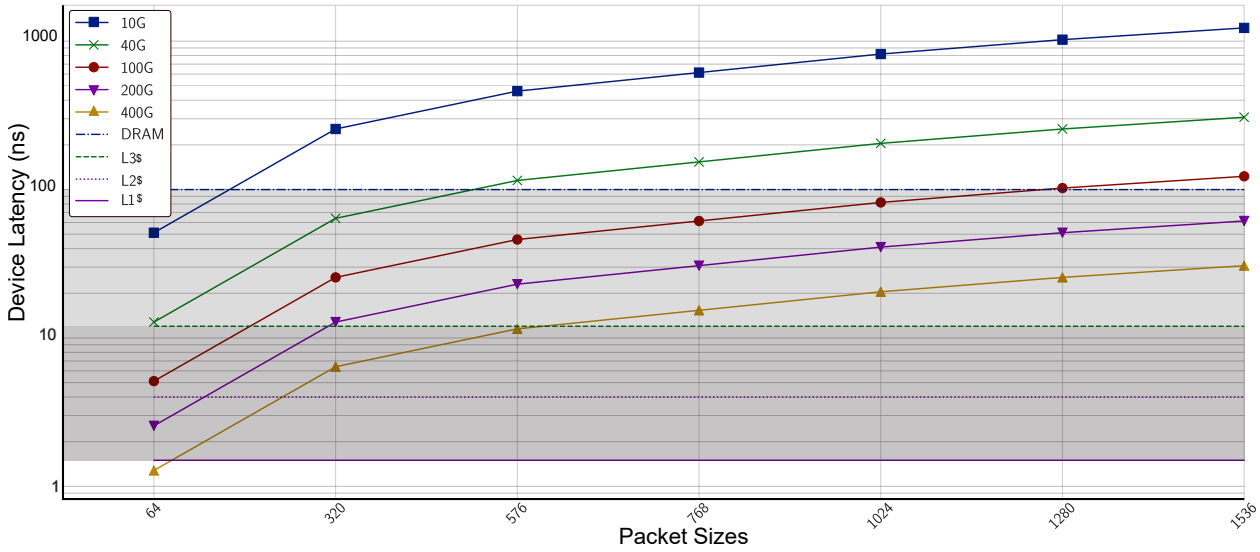

Figure 1: **Evolution of link speeds and interpacket gaps compared to CPU memory hierarchy latencies. The top region represents the regime where the network was slower than DRAM. In the middle region, networks are faster then DRAM but slower than cache. In the darkest region, minimum sized packets exceed L3 cache latency for all network speeds above 100G.**

2.3 What are the advantages and barriers?

Stubbornly slow DRAM latency: With 100G link speeds becoming commodity and 400G on the horizon, the network and architecture community are at a tipping point. 100 Gb/s links transmit minimum sized packets at a rate of one every 5ns while a DRAM access is more than an order of magnitude higher. Even if we consider receive side scaling among a generous allocation of 12 physical cores, the processor would only then have 60ns to process that packet while DRAM latencies are stuck at 100ns. If each packet incurs a cache miss (i.e. a full memory access), queues in the NIC would fill up and packets would drop. While RAM typically supports higher-performance "burst modes", packet data is not a good fit for the spatial locality required to take advantage of burst mode RAM.

The result of DRAM-based packet drops is increased congestion, a higher number of retransmits, low overall throughput, and a significant hit to application-observed latency. At 400G this becomes a problem even for MTU sized packets, as packet rates in that regime will be transmitted at a rate of once every 30ns.

PCI Express latency: While RAM latency is approximately 100ns, the latency of arbitrating and crossing the PCI Express bus could be as high as 800ns/round-trip [19]. Both PCIe and DRAM have one common property, each of them force the CPU to access off-chip data. While both seem like unavoidable hardware issues that systems designers have no leverage over we argue that this is not the case. Considering the NIC as a "peripheral", no different than a mouse or keyboard, is deeply flawed in modern datacenters. Indeed, at 100 Gb/s and 400 Gb/s, the NIC is a major pipe in much the same way that memory bandwidth is a major pipe into and out of the processor. Co-packaging the NIC into the processor itself is likely necessary for realizing the vision of an endhost that can process network traffic at or near the underlying propagation delay of the channel. While a seemingly drastic redesign of modern server architectures, the idea of a NIC integrated with the processor appeared a decade ago as part of Sun Microsystem's Niagra 2 processor [21].

DDIO: Network latencies have moved beyond the *DRAM regime* into the *cache regime*. Some techniques exist to help us navigate this new era. Technologies such as DDIO place packet data directly inside LLC cache and bypass DRAM entirely on both writes and reads. This avoids expensive DMAs and allows network stacks to operate directly from cache without needing to perform memory access with each packet transfer. While a good step forward, DDIO suffers from a number of limitations, including its limited use of cache (about 10% of the LLC cache size can be used by DDIO [14]), as well as the inability to control the placement of data into the LLC using DDIO. This lack of control poses a major problem for holistically managing the cache as a critical resource in supporting ultra-low latency endhosts.

Datacenter Cost: Higher speed networks require a requisitely larger fraction of cores to spread packet processing tasks across. The higher the core count, the larger the available interpacket gap to tolerate software and hardware inefficiencies. This adds to the TCO of the datacenter as faster links will also require more costs in infrastruc-

ture upgrades in the form of a higher physical core count. Table 2 shows total core count needed to achieve line rate if each packet made an access to SRAM versus DRAM at varying speeds. At 400G, line rate packet processing with a single DRAM access will take 79 **physical** cores. Keeping the data in the L3 cache can reduce this down to 19 cores. This 4x increase in core count also accounts for a 1.5-2x increase in cost based on the available selection of modern Intel Xeon processors [13]. More control over the L2 and L1 caches can bring this core count down further as the access latencies here are even lower than the L3 cache. Higher link speeds could then be accommodated with no higher core counts than we have today.

2.4 How do applications benefit?

Some network applications have footprints or working sets that fit within the cache. Such applications often set throughput as the primary performance metric with the goal of operating at line rate. These applications include software forwarding and routing [31], firewall and network intrusion detection [30], and network function virtualization. Such applications essentially process packets through a data structure at line rate, and these data structures can be maintained in cache. Panda et. al. shows that for a network function performing longest prefix matching (LPM) throughput drops linearly with the number of main memory accesses per packet [25]. In other cases, components of a larger distributed system act as serialization points, such as high-throughput sequencers in fault-tolerant distributed systems [3]. In these cases, the throughput of the entire system depends on the throughput of the component.

Other network applications will benefit from the sub-microsecond latencies that CacheCloud can provide. The core of such applications is some form of messaging or remote procedure call which are becoming increasingly common in for datacenter applications[4]. The performance of cache and acceleration systems like memcached also fundamentally depends upon message latency since in essence they are just transporting data with minimal computation. Latency of message exchanges is at the heart of fault-tolerant distributed systems.

For these applications, it is unlikely that all of their data structures will fit in the cache (e.g., memcached benefits are tied to the capacity of main memory). In these cases, CacheCloud can accelerate the "hot" portion of the workload, whether it is frequently accessed data in memcached, the most common RPC routines, or the core message exchanges in distributed system protocols. In all of these cases applications can benefit greatly from support for explicitly managing where in the memory hierarchy their data resides.

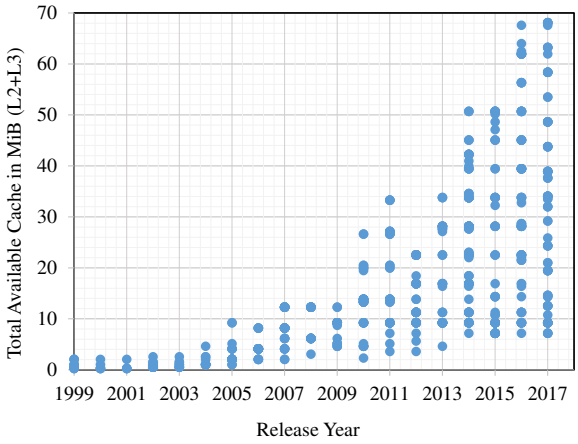

Figure 2: **Total L2+L3 cache (SRAM) available in 700 Xeon processors. Specs from Intel datasheets.**

3 Networking General Purpose SRAM: The Case for CacheCloud

As physical and technilogical constraints bring an end to CPU and DRAM latency scaling, we propose one solution to side step these issues: explicitely expose the rest of the memory hierarchy to the programmer. Like main memory, SRAM (used in CPU caches) latencies have also remained constant for years, albeit at two orders of magnitude lower latency than DRAM. While SRAM has two major drawbacks, small sizes and limited control, current hardware trends may allow for the networking of all available general purpose server SRAM for completely in-cache distributed systems. Inspired by RAMCloud [24], we call this SRAM cluster "CacheCloud".

The cost per bit for SRAM is two orders of magnitude more than DRAM [6]. This is an incredibly expensive premium for a resource that users have no explicit control over. In spite of this, cache sizes have constantly grown. Figure 2 shows a survey of 700 Xeon processors since 1997 and their combined L2+L3 cache sizes. Over the last decade there has been an exponential growth in average cache size of available processors. While SRAM will not replace DRAM for applications in general, we argue that networking all available general purpose server SRAM opens the doors for ultra low-latency and low variance packet processing. For example, if all the caches in a 1000 node cluster can be addressed as a single unit, 100MiB of SRAM turns into 1000GiB of SRAM — something not possible in a single server due to energy, area, and technology constraints.

To enable networked SRAMs in today's datacenters, general purpose processors must support fine grain software management of traditionally black box hardware policies. This includes control over policies such as cache

replacement, consistency and coherence protocols, and prefetching. We have seen some progress on this in recently years. There has been an explosion of ISA extensions for Intel [11] and ARM [2] exposing some control over low level policies. Intel processors include instructions for memory prefetching (PREFETCHW), optimized cache flush (CFLUSHOPT), cache partitioning (CAT), and cache line write-back (CLWB). While ARM provides instructions such as data cache invalidate by set/way (DC ISC) and data cache clean by set/way (DC CSW). In addition there has been recent pushes in the architecture community for improving determinism in general purpose memory hierarchies [23]. For both systems designers and architects this opens the door for new explorations of bespoke hardware policies for network based application. For example, are current hardware prefetchers appropriate for network applications? Can we predict cache misses before they happen and reroute packets into locations where data is resident in cache? Additionally, can we redesign cache replacement policies from being server oriented to cluster oriented without complete hardware specialization?

We believe that this trend of increasing the control of CPU policies will enable a new class of research focusing on these questions and allow CacheCloud clusters to become a reality.

4 Challenges and open problems

A number of open problems remain to fully realize the CacheCloud vision.

Software Challenges: There are two major software challenges to leverage global cache control: (1) a Distributed Cache Scheduler (DSC), and (2) a progamming API. Enabling a distributed SRAM cluster requires nodes to maintain a global view of the current state of the system and make decisions based on this state. This itself has challenges for maintaining consistency across the cluster and ensuring that data is being placed strategically to avoid going to DRAM. The DSC may be implemented as a traditional cluster scheduler such as Mesos [12] or Yarn [35], or functionality can be pushed to network devices. Systems like Eris [20] show that programmable switches may be one avenue to maintain global state for consensus while maintaining very high throughput.

The second, and possibly most important challenge, is programmability. It is an open question about how much of CacheCloud should be exposed to the application developers and how much policy control they should have to take advantage of CacheCloud mechanisms. One question is whether applications must be forced to conform to CacheCloud primitives that explicitly place data in certain cache lines, or whether the burden should fall on the DSC to provide high level guarantees such as maintaining line rate processing for each application on the box.

Hardware Challenges: Beyond software improvements, the hardware challenges for CacheCloud require collaboration from processor and interconnect vendors. To this end, the systems and architecture communities must work together to decide which hardware components are most appropriate to expose to enable next generation high speed networks. FlexNIC is one proposal that offloads some of the packet processing into the NIC SRAM and steers packets into cores based on coarse grain application level information [17] but does not provide interface over the end-host cache. There has been work on QoS-aware memory and prefetching systems at the hardware [18, 9] and software level [22, 5] but most require specialized solutions or focus on using cache instructions to perform isolation for noisy neighbors. We observe that in contrast to the architecture community, which is moving towards specialization with devices such as Google's TPU [15], the network community with trends such as NFVs and programmable switches and NICs are moving towards flexibility. Closing the performance gap between these flexible solutions and their specialized counter-parts (e.g., hardware middleboxes and NFVs) requires hardware flexibility at all levels, including the end-hosts, to take advantage of programmable network hardware.

The second major hardware challenge is the PCIe bus. PCIe as a technology is built for batch transfers, not necessarily low latency. Alternative interconnects could help eliminate the latency overhead of this bus. Another option is directly co-packaging the NIC with the processor or exploit near data computing [26] or in-network computing.

5 Conclusion

As network speeds march beyond 400G it presents both an opportunity and a challenge that the network, OS, and architecture community must approach together. High speed links require us to drastically change our view of the memory hierarchy to sustain line rates and obtain low latency. Ultimately, the unbalanced scaling that has plagued systems designers must be reconciled with through hardware, software, and network co-design. To this end, CacheCloud is one proposal that considers such a design. Only with all three components working in lock step will we achieve latencies that truly match the speed of light.

Acknowledgments

We would like to thanks Luis Vega for insightful discussions on this work. This work funded in part by the National Science Foundation (CNS-1564185, CNS-1629973, and CNS-1553490).

References

[1] Mohammad Al-Fares, Alex Loukissas, and Amin Vahdat. A scalable, commodity, data center network architecture. In *Proceedings of the ACM SIGCOMM Conference*, Seattle, WA, August 2008.

[2] ARM. ARM Architecture Reference Manual ARMv8. `https://static.docs.arm.com/ddi0487/ca/DDI0487C_a_armv8_arm.pdf`, 2017.

[3] Mahesh Balakrishnan, Dahlia Malkhi, John D. Davis, Vijayan Prabhakaran, Michael Wei, and Ted Wobber. CORFU: A Distributed Shared Log. *ACM Transactions on Computer Systems (TOCS)*, 31(4), 2013.

[4] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the killer microseconds. *Commun. ACM*, 60(4):48–54, March 2017.

[5] Henry Cook, Miquel Moreto, Sarah Bird, Khanh Dao, David A. Patterson, and Krste Asanovic. A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 308–319, New York, NY, USA, 2013. ACM.

[6] Digi-key integrated circuits: Memory. `https://goo.gl/k9oEBW`.

[7] Doug Coleman. Optical trends in the data center. https://www.bicsi.org/uploadedFiles/BICSI

[8] DPDK. Data plane development kit. `https://dpdk.org/`.

[9] Eiman Ebrahimi, Onur Mutlu, Chang Joo Lee, and Yale N. Patt. Coordinated control of multiple prefetchers in multi-core systems. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 316–326, New York, NY, USA, 2009. ACM.

[10] John L. Gustafson. Reevaluating Amdahl's Law. *Commun. ACM*, 31(5):532–533, May 1988.

[11] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer. Cache QoS: From concept to reality in the Intel Xeon processor E5-2600 v3 product family. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 657–668, March 2016.

[12] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[13] Intel Xeon scalable processors. `https://ark.intel.com/products/series/125191/Intel-Xeon-Scalable-Processors`.

[14] Intel Corporation. Intel Data Direct I/O technology (Intel DDIO): A Primer. `http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/data-direct-i-o-technology-brief.pdf`, 2012.

[15] Norman P. Jouppi *et al.* In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[16] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the ACM SIGCOMM 2017 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2017.

[17] Antoine Kaufmann, Simon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 67–81, New York, NY, USA, 2016. ACM.

[18] Chang Joo Lee, Veynu Narasiman, Eiman Ebrahimi, Onur Mutlu, and Yale N Patt. DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. 2010.

[19] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. Kv-direct: High-performance in-memory key-value store with programmable NIC. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 137–152, New York, NY, USA, 2017. ACM.

[20] Jialin Li, Ellis Michael, and Dan R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 104–120, New York, NY, USA, 2017. ACM.

[21] G. Liao and L. Bhuyan. Performance measurement of an integrated NIC architecture with 10GbE. In *2009 17th IEEE Symposium on High Performance Interconnects*, pages 52–59, Aug 2009.

[22] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 450–462, New York, NY, USA, 2015. ACM.

[23] Mark Rutland (ARM). Stale data, or how we (mis-)manage modern caches. `https://goo.gl/WtwfHk`, 2016.

[24] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. The case for RAMClouds: Scalable high-performance storage entirely in DRAM. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, January 2010.

[25] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of nfv. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, Berkeley, CA, USA, 2016. USENIX Association.

[26] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, March 1997.

[27] David A. Patterson. Latency lags bandwidth. *Commun. ACM*, 47(10):71–75, October 2004.

[28] Removing roadblocks on the path to 400G and beyond. `https://goo.gl/ZyE6bd`.

[29] Luigi Rizzo. Netmap: A novel framework for fast packet i/o. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[30] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[31] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24, New York, NY, USA, 2012. ACM.

[32] Ankit Singla, Balakrishnan Chandrasekaran, P. Brighten Godfrey, and Bruce Maggs. The Internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 1:1–1:7, New York, NY, USA, 2014. ACM.

[33] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, 2012.

[34] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards Optimal-Performance Datacenters. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies (CoNEXT'16)*, pages 205–219, 2016.

[35] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[36] Verizon marks milestone with successful 400G technology trial. `https://goo.gl/y54Jaa`.