Towards Minimalistic, Virtualized Content Caches with Minicache

Simon Kuenzer, Joao Martins, Mohamed Ahmed, Felipe Huici
NEC Europe Ltd.

ABSTRACT

Video comprises the majority of traffic on the Internet today, and most of it is delivered via Content Delivery Networks (CDNs) whose performance depends, to a large extent, on being able to deploy a (sometimes rather large) set of distributed content caches at different networks and geographical locations. Recently, ISPs have started deploying *micro datacenters* close to customers, giving the possibility to third parties to rent out this equipment.

While such pay-on-demand infrastructure would allow CDNs to dynamically expand their capacity and improve their efficiency, a high performance, virtualized content cache would be needed for multiple tenants to be able to share such facilities. Towards this end we introduce Minicache, a Xen-based virtualized content cache prototype. A Minicache virtual machine has a small memory footprint (as small as 5MB in size), can boot in as little as 30 milliseconds, and can fill up a 10Gb pipe using data retrieved from storage devices.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*; D.4.8 [**Software**]: Operating Systems—*Performance*

Keywords

Virtualized Content Caches; CDN; Virtualization; Block I/O Performance; Video Streaming

1. INTRODUCTION

While the Internet is used to carry packets for a wide range of applications and services, the past years have seen video traffic rapidly expand, to the point where today it comprises the largest portion of all traffic on the network. A recent report states that; consumer Internet video traffic will make up 69% of all consumer Internet traffic by 2017, up from 57% in 2012 [4]. Within this, real-time entertainment dominates, with Netflix accounting for as much as 33% of peak period downstream traffic in North America [20].

The large majority of this traffic is delivered via Content Delivery Networks (CDNs), that are expected to transport almost two-thirds of all video traffic by 2017 [4]. CDNs optimize video delivery by deploying so-called content caches in different networks, often geographical dispersed, so that user requests can be handled locally by nearby caches. For instance Akamai, the market leader, claims to deliver more than 20% of all Web traffic worldwide [15], and does so by deploying more than 100,000 servers in more than 1,800 locations across nearly 1,000 networks [7].

One of the reasons behind such large-scale distribution is that TCP requires receiver acknowledgments for every window of data sent, meaning that throughput is inversely related to network latency or RTT. The end result is that the distance between the server and end user can become the overriding bottleneck in delivering good video quality [23]; for instance, a latency of 96 milliseconds would result in a rather poor rate of 0.4 Mb/s [15]. Other factors such as sub-optimal BGP routing, so that two locations within a continent are routed through another only exacerbate the problem [17].

While Akamai is the largest player many others exist, ranging from traditional CDNs to ISP-operated CDNs, content provider CDNs, free CDNs and even peer-to-peer CDNs [25]. Their business models and infrastructures vary, but what they all have in common is that they would benefit from having a presence, even if small, at a large number of distributed points in networks, such as the Points of Presence (PoPs) that operators deploy at aggregation points in their networks. Proof of this is the fact that Akamai has already started forming alliances with a number of large ISPs, including AT&T, Orange, Swisscom and KT [7].

Beyond such two-way agreements, more recently ISPs have started deploying small datacenters called *micro datacenters* at PoPs and to allow third parties access to these infrastructures [7]. The availability of such "pay-on-demand" infrastructure would allow CDNs to dynamically expand their capacity and reach, without having to go through the expensive and time-consuming process of deploying hardware and facilities. Such an arrangement would be especially beneficial to smaller players in the market who cannot afford large up-front investments.

In order for such arrangements to work, virtualization would be required to enable the safe sharing of common infrastructure resources (in the form of content caches) across different CDN operators. In addition, virtual machines have

HotMiddlebox'13, December 9, 2013, Santa Barbara, CA, USA.
Copyright 2013 ACM 978-1-4503-2574-5/13/12 ...$15.00.
http://dx.doi.org/10.1145/2535828.2535832.

the potential to be quickly instantiated, enabling a CDN to adapt to changing load conditions that can arise from events such as flash crowds. In fact, a recent study of three major CDNs concluded that quality, as measured by metrics such as rebuffering ratio, video startup time and video start failure, varies considerably across time and space, indicating the need for providers to dynamically choose different nodes for different clients [10]. Micro datacenters, along with quickly instantiable virtual cache nodes, might go along way towards improving CDN performance.

In light of the considerations discussed so far, we introduce Minicache, a virtual cache node prototype. Minicache is a tiny single-purpose Xen virtual machine built on top of MiniOS, a minimalistic, para-virtualized OS available with the Xen sources. Our contributions consist of (1) a measurement study of Xen's block I/O sub-systems and its bottlenecks, (2) a prototype implementation of Minicache that packetizes data retrieved from block devices and sends the resulting packets out to a NIC at rates of 10Gb/s, and (3) a preliminary implementation and evaluation of an HTTP server on top of MiniOS that we will eventually merge with Minicache.

Because it is built on top of a minimalistic OS, Minicache's virtual machines can be instantiated in 30 milliseconds and have a memory footprint of only 5 MB [*], allowing it to quickly scale out to make use of additional resources on a server such as CPU cores, SSD drives and NIC ports without need for content replication; for instance, 64-core AMD servers are now inexpensive, in the order of $3,000, and other companies such as Tilera are pushing towards hundreds of cores in a single chip [26].

Such small boot times matter: a study of flash crowds and their effects on CDNs shows that even having boot times in the order of seconds can result in over-subscription, and that services such as Amazon's EC2 can only spin up VMs in one second at best [24]. Being to able to react this quickly to load also means that content caches can be run at higher utilization levels than usual.

The rest of the paper is organized as follows. In section 2 we present results from a performance analysis that points out potential bottlenecks in Xen's block I/O sub-system. Section 3 then describes Minicache's design, followed by a performance evaluation of it in section 4. Finally, in section 5 we present related work and section 6 discusses future work and concludes.

2. XEN BLOCK I/O PERFORMANCE ANALYSIS

As mentioned in the previous section, Minicache is based on the Xen hypervisor [2]. We chose Xen for a number of reasons. First, the fact that its guests are para-virtualized means that we do not incur overheads from operations such as VM exits or instruction emulations. Second, the Xen sources come with a built-in minimalistic, para-virtualized OS called MiniOS that eliminates a number of expensive operations such as context switches or system calls. Finally, this choice allows us to leverage previous results that optimized Xen's I/O network pipe [11].

Before beginning our work with Minicache, we wanted to measure any potential performance bottlenecks in Xen's block I/O sub-system. In this section we present results from such an analysis, but first we provide a brief background on Xen, how it handles I/O, and MiniOS.

[*]Please refer to [11] for details on how these figures are achieved.

2.1 Background

Xen is a Type-1 or bare-metal hypervisor that takes care of tasks such as memory management and CPU scheduling among the available virtual machines (VMs). Further, Xen has a special VM called *domain0* or *dom0* that runs a standard OS such as Linux or FreeBSD and has special privileges that allow it to, among other things, create and destroy other VMs called *domUs* (user or guest domains).

In terms of I/O, Xen uses a *split driver* model. Under this model, a driver domain, typically dom0, implements a back-end driver that talks to the actual physical drivers on one end (e.g., `ixgbe` in the case of an Intel 10Gb card) and implements a common API on the other. Guest domains, such as a Linux VM, then implement a front-end driver that conforms to this API. This allows guests to have (indirect) access to device drivers that might not be available (e.g., MiniOS, which we briefly describe below, does not have network driver support).

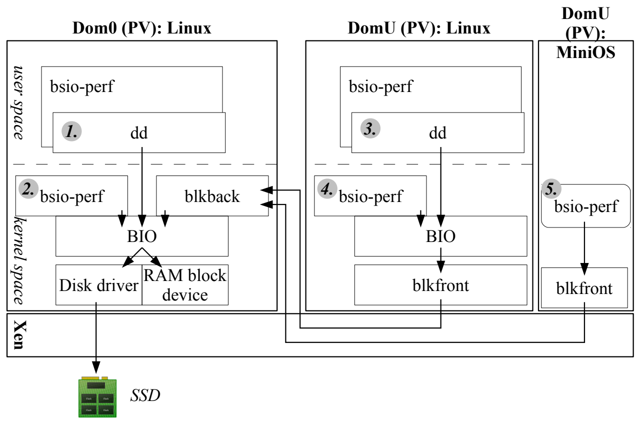
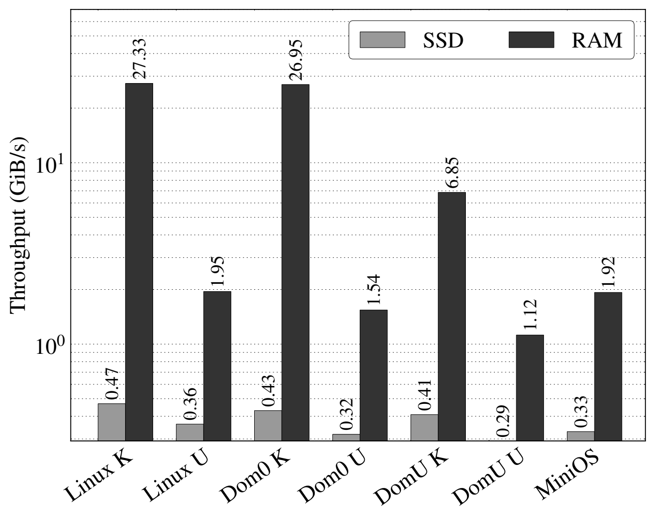


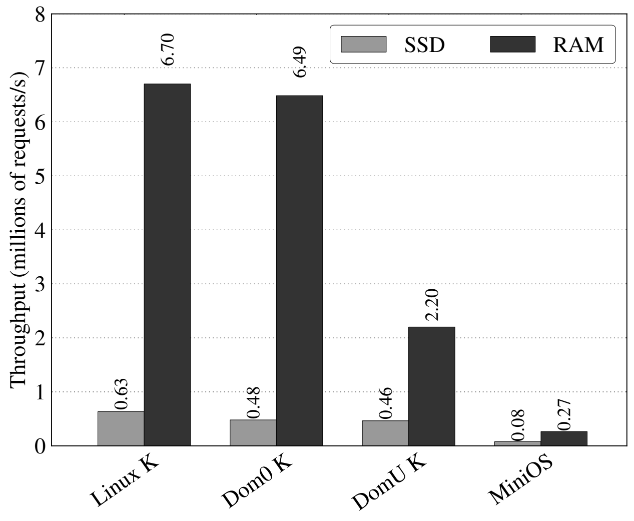
Figure 1: Xen block I/O performance evaluation setup. The circled numbers represent different measurement points taken with our *bsio-perf* tool.

Each type of device gets its own pair of back-end and front-end drivers: for instance, for network I/O, the driver domain implements a `netback` driver and guests a `netfront` one. Likewise, block operations are done via `blkback` and `blkfront` drivers. The `blkback` driver then plugs into Linux's BIO interface which itself connects to either disk drivers (e.g., for SATA disks) or to a RAM block device (for RAM disks); figure 1 illustrates the connections from a Linux domU via its `blkfront` and through dom0's `blkback`, BIO and block device drivers.

Finally, it is worth mentioning that Minicache is based on MiniOS. MiniOS comes with the Xen sources and provides the minimum functionality needed to run as a Xen virtual machine: `netfront` and `blkfront` drivers, as well as a Xen bus and store drivers to send and receive control messages. It implements a simple co-operative scheduler, so incurs no preemption or other scheduling costs, and has a single memory space, eliminating costly system calls. Note that this means improved performance but also that we cannot easily run multiple tenants within the same Minicache VM. How-



(a) Throughput in GiB/s



(b) Throughput in requests/s

Figure 2: Block I/O performance results using the *bsio-perf* tool for different scenarios. A "U" label means user-space, "K" kernel space. Both dom0 and domU labels refer to a Linux virtual machine. MiniOS does not have "K/U" labels since it has a single memory space.

ever, this is not a problem: new tenants can be easily (and quickly) assigned a new Minicache VM, and we can run up to hundreds of them on the same system [11].

2.2 Experimental Setup

We evaluated the performance of Xen's block I/O subsystem with *bsio-perf*, a small tool we developed. In order to get a wide range of measurement points *bsio-perf* consists of (1) a wrapper around `dd` when running in Linux user-level (scenarios 1 and 3 in figure 1); (2) a kernel module when running in Linux kernel space (scenarios 2 and 4); (3) a built-in application when running on top of MiniOS (scenario 5). We also include two baseline scenarios without any virtualization: in-kernel Linux and user-space Linux (not shown in the figure). The tool measures performance by reading a number of chunks of a certain size (set to 32 MiB in our experiments) from a disk for different block sizes and keeping track of both throughput and the number of requests per second.

All measurements were conducted on an inexpensive (about $1,500), single-socket Intel Xeon E3-1220 system (4 cores at 3.1 GHz) with 16 GB of DDR3-ECC RAM, a Crucial m4 128 GB SATA SSD[5], Linux 3.9.6, and Xen 4.2.0. We also use a RAM drive to investigate bottlenecks that are only visible are rates beyond those that the SSD can attain. We focus our tests on read performance since a cache node is likely to experience read-intensive workloads, but also to prevent performance degradation on the SSD drive, as would be the case with write tests. Finally, we use 16KB blocks when doing read requests, as this yields the best performance and so makes it simpler to discover and understand bottlenecks.

2.3 Measurements Results

Figure 2a shows throughput results for the different scenarios. As a baseline, we begin by measuring the performance of astandard Linux system with no virtualization (Linux "K" for kernel and "U" for user-space). The first thing to notice is that for the SDD in-kernel case we obtain a throughput of 470 MB/s, close to the drive's theoretical maximum. The RAM drive, as expected, is able to push much higher rates, reaching roughly 27 GB/s. This drive also enables us to see the massive performance hit incurred when running our tool from user-space (essentially `dd`), where the rate collapses to 1.95 GB/s. This is to be expected: running the test from user-space not only results in costly system calls, but also goes through additional layers including the file system. Given the SSD's lower maximum rate, the rate drop when used from user-space is lower, though still noticeable.

The next set of bars shows results from running our tool from within Xen's dom0. The main take-away here is that these numbers do not overly differ from those from the Linux system, meaning that the Xen hypervisor introduces only slight overhead. The next set measures performance from a Xen Linux guest domain ("DomU K" and "DomU U"). The reduced rate of 6.85 GB/s is due to the cost of going through Xen's `blkback` and `blkfront` drivers (refer back to figure 1). Note that this performance hit is not intrinsic to having a hypervisor (the previous set of bars ruled that out) nor to the split driver model, but rather the results of unoptimized drivers; previous work [11] showed that high rates can be achieved by improving such (in that case network) drivers.

The final bars show MiniOS' performance. The rate of 1.92 GB/s points to the fact that MiniOS' `blkfront` driver yields much poorer performance than the one in the Linux virtual machine (6.85 GB/s). This is unsurprising given that MiniOS drivers where written to be functional, but not necessarily to perform well [11].

To provide a different view, figure 2b plots measurements in terms of requests per second, where a request means a block read. Instead of using 16KB blocks, we used 512B blocks here in order to raise the pressure to the I/O system in handling requests. As expected, throughput is inversely correlated with the number of requests per second that the different setups can handle. The worst culprit is the MiniOS `blkfront` driver; despite this, in section 4 we will see that, because transfer/block sizes are large, these rates are still good enough to fill a 10Gb network pipe.

3. MINICACHE DESIGN

Our current Minicache prototype is built on top of MiniOS and encapsulates block data read via the SSD and RAM disks into UDP packets (figure 3). In short, Minicache's main loop submits arbitrary block I/O read requests from the *blkfront* driver and splits the block read into multiple packet buffers. Each buffer is encapsulated in Ethernet/IP/UDP headers before being sent out to the optimized, Netmap-based `netfront` driver introduced in [11].

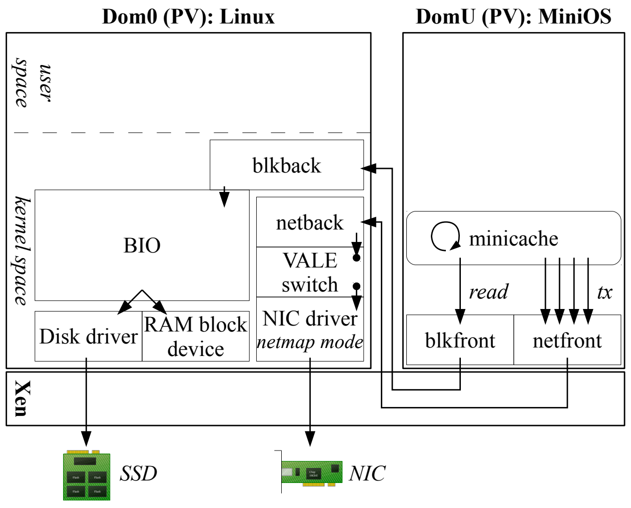


Figure 3: Minicache architecture.

In addition, Minicache has a pool memory allocator used for temporarily storing block data as well as packet buffers about to be sent out. Pool memory is allocated before any processing is needed in order to avoid allocation costs, including memory alignment operations, when processing requests. In addition, we use a ring structure to keep track of references to unused memory objects. If objects are *picked* from the pool, the object's reference gets dequeued and returned to the caller; the opposite operation (*put*) enqueues a reference back onto the ring.

Figure 4 illustrates the main steps involved in Minicache's main processing loop:

- First, Minicache polls the `blkfront` driver to see if there are any completed I/O requests. If so, we mark the request as completed (see third step) and carry out a few clean-up tasks.

- After that, we pick new receive buffers from the memory pool and submit new block I/O requests to MiniOS' `blkfront`.

- Once submitted, we begin packetizing the block data received in the first step. More specifically, Minicache splits one data block into multiple 1070-byte packets (1024 bytes for payload and 42 bytes for the Ethernet, IPv4 and UDP headers)[†]. Each packet is created from scratch using a packet buffer picked from the packet buffer memory pool. After the block data copy and packet header writing is done, Minicache puts the block buffer back in its pool.

- Finally, we pass the packet buffers to the optimized `netfront` driver implemented in [11]. Once sent, we return the buffers to their memory pool.

[†]We chose 1024 bytes as payload because this is a common divisor of the block sizes we use in our system.

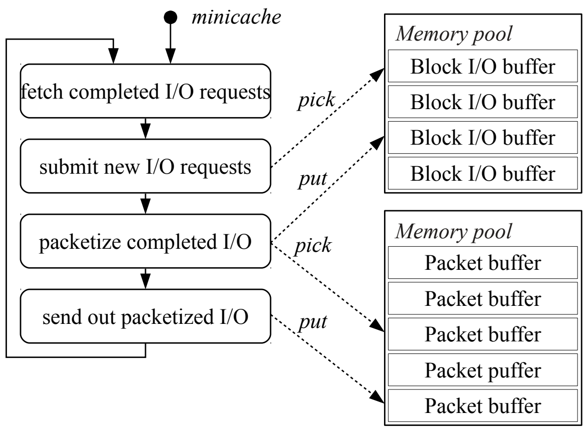


Figure 4: Minicache's main loop and memory allocation scheme.

4. EVALUATION

We evaluated Minicache's performance by measuring the packet rate generated when reading blocks of different sizes from either the SSD or the RAM drive and packetizing the data. We performed the experiments on the same server described in section 2.2, this time connected to another server of similar specs running netmap [18] to receive packets and measure rates. Both servers have Intel X520-T2 cards linked via a direct cable. In addition, the Minicache server's `netback` driver is connected to a modified, netmap-enabled `ixgbe` driver via the VALE software switch [19] instead of Xen's standard Open vSwitch.

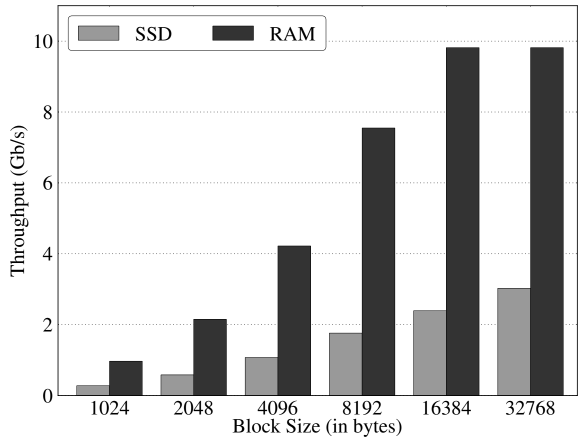


Figure 5: Minicache's performance.

The results are shown in figure 5. For 16KB blocks and larger and using the RAM drive Minicache is able to fill up an entire 10Gb pipe. For the SSD the rate is expectedly lower, limited by the SSD's maximum rate. For smaller block sizes we can start to see the impact of MiniOS' unoptimized `blkfront` driver (recall figure 2b), especially in the case of the RAM drive.

HTTP Server: Towards a more realistic content cache, we started developing an HTTP server on top of MiniOS which we will eventually integrate with Minicache. To measure its performance we relied on the `wrk` benchmark tool [8]. For now, the server can handle a rate of 2,000 requests/s on a single CPU core. For comparison, we ran a simple test using a node.js [14] HTTP server on dom0 and obtained rates of 4K requests/s; we are currently in the process of improving our server's performance.

5. RELATED WORK

Storage virtualization is typically used in the elastic cloud offerings of many organizations. To name a few, the products from Amazon (EC2), OnApp (OnApp Cloud) and VMware (vCloud) build cloud infrastructures to provide software as a service. In contrast, Minicache aims to address the problem of virtualizing caches for content distribution networks. Content cache servers are typically specialist devices (appliances), built on general-purpose hardware (Netflix Open Connect [13], OnApp Edge server [16], Akamai Aura [1], and EdgeCast [6]), with standard OSes, and can be further accelerated using special-purpose devices such as NetApp's FlexCache [12] to provide in-memory caching. Unlike the above, Minicache provides a single purpose OS for content caching. To the best of our knowledge, Minicache is the first attempt at a virtualized and minimalist content cache OS.

With regard to the performance of storage under virtualization, Shafer [21] finds that when using IDE disks, the (para)virtualization cost can lead to between 49 - 30% of the native system performance. Similarly, we find that with SSDs, the cost is between 39 - 30% (again, due to unoptimized drivers). Since typically guest VM filesystems run on top of the host's filesystem, the extra layers of indirection add cost; Le et.al. [9] study this cost and find that it can be as high as 67% of the native performance. Minicache will not suffer this cost since we aim to hook its filesystem directly onto the block devices running in Dom0. Aside from the block I/O and the filesystem, I/O scheduling[28, 3] and the mounted image format (QCOW, VMDK and etc) [22] can have a significant impact on the performance of virtualized storage. Since Minicache is a non-SMP OS with a cooperative scheduler and directly interacts with block devices instead of flat disk images, these costs are mitigated.

Finally, towards improving Xen block performance, recent efforts by the Xen community have shown gains from using persistent grants in the Linux `blkback`/`blkfront` drivers [27]. This is a well-known technique for improving Xen split driver performance and one of the mechanisms we will apply to improve MiniOS' `blkfront` driver.

6. CONCLUSION AND FUTURE WORK

We introduced a performance evaluation of the Xen block I/O subsystem and its bottlenecks, as well as a proof-of-concept implementation of Minicache, a small (5MB), quickly instantiated (30 milliseconds) Xen-based virtual machine aimed at content caching. Minicache is able to packetize data read from storage devices and to fill up a 10Gb pipe with the resulting packets.

Minicache is at an early stage and needs further work to bring it closer to being a content cache. First, we will look into optimizing the MiniOS `blkfront` driver so that it does not become a bottleneck for smaller block sizes. In addition, we need to introduce a filesystem on top of the `blkfront` driver since MiniOS does not provide one. Likely we will rely on a straw-man "filesystem" based on a hash table mapping content names to block ids in order to test raw performance, and then move on to a full-fledged filesystem (e.g., EXT2, as this would probably be easier to port to MiniOS than EXT4).

Further, we are currently improving the performance of our HTTP server and will soon begin to integrate with Minicache. Beyond this, we are planning on running experiments with increasing number of Minicache virtual machines and CPU cores to measure how the system would perform when concurrently hosting a potentially large number of CDN tenants.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under Trilogy 2 project, grant agreement number 317756.

8. REFERENCES

[1] Akamai. Aura Licensed CDN. `http://www.akamai.com/html/solutions/aura_licensed_cdn.html`, 2013.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. ACM SOSP, 2003*, New York, NY, USA, 2003. ACM.

[3] David Boutcher and Abhishek Chandra. Does virtualization make disk scheduling passé? *SIGOPS Oper. Syst. Rev.*, 44(1):20–24, March 2010.

[4] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2012-2017. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`, 2012.

[5] Crucial. *Crucial m4 SSD 2.5" Technical Specification*, 2011.

[6] EdgeCast. Carrier CDN Solution. `http://www.edgecast.com/solutions/licensed-cdn/`, 2013.

[7] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing cdn-isp collaboration to the limit. *SIGCOMM Comput. Commun. Rev.*, 43(3):34–44, July 2013.

[8] GitHub. Modern HTTP benchmarking tool. `https://github.com/wg/wrk`, July 2013.

[9] Duy Le, Hai Huang, and Haining Wang. Understanding performance implications of nested file systems in a virtualized environment. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[10] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 359–370, New York, NY, USA, 2012. ACM.

[11] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling fast, dynamic network processing with clickos. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, HotSDN '13, pages 67–72, New York, NY, USA, 2013. ACM.

[12] NetApp. NetApp FlexCache. `http://www.netapp.com/us/products/storage-systems/flash-cache/index.aspx`, 2013.

[13] Netflix. Netflix Open Connect Content Delivery Network. `https://signup.netflix.com/openconnect`, 2013.

[14] Nodejs.org. node.js. `http://nodejs.org`, August 2013.

[15] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[16] OnApp. Edge Server appliance, OnApp CDN Stack. `http://onapp.com/cdn/technology/edge-server/`, 2013.

[17] Hariharan Rahul, Mangesh Kasbekar, Ramesh Sitaraman, and Arthur Berger. Towards Realizing the Performance and Availability Benefits of a Global Overlay Network. In *PAM 2006*, Adelaide, Australia, March 2006.

[18] L. Rizzo. netmap: a novel framework for fast packet I/O. In *Proc. USENIX ATC*, 2012.

[19] L. Rizzo and G. Lettieri. Vale: a switched ethernet for virtual machines. In *Proc. ACM CoNEXT*, December 2012.

[20] Sandvine Inc. Sandvine global internet phenomena report. `http://www.sandvine.com/downloads/documents/Phenomena_2H_2012/Sandvine_Global_Internet_Phenomena_Report_2H_2012.pdf`, 2012.

[21] Jeffrey Shafer. I/o virtualization bottlenecks in cloud computing today. In *Proceedings of the 2nd conference on I/O virtualization*, WIOV'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association.

[22] Chunqiang Tang. Fvd: a high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 18–18, Berkeley, CA, USA, 2011. USENIX Association.

[23] Sipat Triukose, Zhihua Wen, and Michael Rabinovich. Measuring a commercial content delivery network. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 467–476, New York, NY, USA, 2011. ACM.

[24] Patrick Wendell and Michael J. Freedman. Going viral: flash crowds in an open cdn. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 549–558, New York, NY, USA, 2011. ACM.

[25] Wikipedia. Content Delivery Network. `http://en.wikipedia.org/wiki/Content_delivery_network`, 2013.

[26] Wikipedia. Tilera. `http://en.wikipedia.org/wiki/Tilera`, 2013.

[27] Xen Project Community Blog. Improving block protocol scalability with persistent grants. `http://blog.xen.org/index.php/2012/11/23/improving-block-protocol-scalability-with-persistent-grants`, 2013.

[28] Jianyong Zhang, Anand Sivasubramaniam, Qian Wang, Alma Riska, and Erik Riedel. Storage performance virtualization via throughput and latency control. *Trans. Storage*, 2(3):283–308, August 2006.