VOL. 4, NO. 1 – FEBRUARY 2015

VMWARE TECHNICAL JOURNAL

Editors: Rita Tavilla and Curt Kolovson
Guest Editor: Kit Colbert

TABLE OF CONTENTS

Please send notes and comments to vmtj@vmware.com

Previous versions of the journal are available on line at https://labs.vmware.com/vmtj.

Welcome to the latest edition of the VMware Technical Journal (VMTJ), Volume 4, Number 1.

At VMware, we have a very clear and focused corporate strategy: Be the leader in the software-defined data center (SDDC), end-user computing (EUC), and hybrid cloud computing (our VMware vCloud® Air™ service).

This issue of VMTJ contains several papers from our EUC organization, and I am grateful to Kit Colbert for acting as the guest editor for this issue. To quote from his introduction in this issue about the work of our EUC teams:

The EUC team's mission is to enable a secure virtual workspace for work at the speed of life. The reality is that consumerization of IT is bringing more—and more diverse—devices onto company networks. The "one size fits all" one-desktop-per-employee model no longer works. IT now needs to manage a plethora of different devices, enabling rapid delivery of a user's applications and data to all those devices while at the same time ensuring security and compliance. Users, on the other hand, are demanding a seamless, integrated experience. They want information and apps at their fingertips and want to be able to set down one device, pick up another, and start right where they left off. These are some challenging requirements!

This is certainly true. We are in the midst of a major shift in how workers go about their computing tasks in the enterprise, where mobile applications and cloud computing are rapidly becoming the primary modes of computing. Kit's introduction describes the EUC papers in this issue, as well as a paper by Ravi Soundararajan and Shishir Kakaraddi on how social networking concepts can be applied to system performance management.

In addition to the papers from EUC and the Soundararajan/Kakaraddi paper, this issue contains two papers from professors and graduate students from Georgia Tech. The first, "Reducing Cache-Associated Context-Switch Performance Penalty Using Elastic Time Slicing" by Jammula et al., describes a novel hardware/software approach for implementing variable time slicing to minimize the context-switch overhead associated with cache-warmup slowdowns that can impact certain workloads, particularly in virtualized environments. The second, "FlashStream: A Multitiered Storage Architecture in Data Centers for Adaptive HTTP Streaming" by Moonkyung Ryu and Professor Umakishore Ramachandran, describes a design for a storage system that is optimized for video streaming. This paper is an expanded version of a paper that appeared in ACM Multimedia 2013.

We take great pride in the work of our talented engineers, and we appreciate the excellent work and significant contributions of our colleagues in academia. As always, we welcome your comments on this issue of the VMware Technical Journal.

Curt Kolovson
Sr. Staff Research Scientist
VMware Academic Program (VMAP)

FlashStream: A Multitiered Storage Architecture in Data Centers for Adaptive HTTP Streaming[1]

Moonkyung Ryu
Georgia Institute of Technology
mkryu@google.com

Umakishore Ramachandran
Georgia Institute of Technology
rama@cc.gatech.edu

Abstract

Video streaming on the Internet is popular, and the need to store and stream video content using content distribution networks (CDNs) is continually on the rise thanks to services such as YouTube, Netflix, and Hulu. Adaptive HTTP streaming (AHS) using the deployed CDN infrastructure has become the de facto standard for meeting the increasing demand for video streaming on the Internet. The storage architecture that is used for storing and streaming the video content is the focus of this study. NAND flash memory solid-state drive (SSD) is a promising storage technology for building a high-performance and cost-effective video streaming system when it is used as an intermediate-level cache in a multitiered storage hierarchy. More recently, multitiered storage systems (incorporating SSDs) such as Oracle's ZFS and Facebook's Flashcache offer an alternative to disk-based storage systems for enterprise applications. Both of these systems use the SSD as a cache between the DRAM and the hard disk. The thesis of our work is that the current state of the art in multitiered storage systems, architected for general-purpose enterprise workloads, does not cater to the unique needs of AHS. We have conducted a thorough analysis to understand the reasons for the poor performance of these two systems. To overcome the limitations of those systems, we present *FlashStream*, a multitiered storage architecture that addresses the unique needs of AHS. The key architectural elements of FlashStream include optimal write granularity to overcome the write-amplification effect of flash memory SSDs and a quality-of-service (QoS)–sensitive caching strategy that monitors the activity of the flash memory SSDs to ensure that video streaming performance is not hampered by the caching activity. We implemented FlashStream and experimentally compared it with ZFS and Flashcache for AHS workloads. We show that FlashStream outperforms both of these systems for the same hardware configuration. Specifically, it is better by a factor of two than its nearest competitor, ZFS. In addition, we compared FlashStream with a traditional two-level storage architecture (DRAM + hard disk drives [HDDs]) and show that, for the same investment cost, FlashStream provides 33% better performance and 94% better energy efficiency.

1. Introduction

Video streaming is a killer application for the Internet. Video traffic is growing fast and becoming a dominant fraction of Internet traffic. It is expected that 90% of Internet traffic will be video in 2017 [22]. Such explosive growth of video traffic is attributable to the growing popularity of video services such as YouTube, Netflix, and Hulu. YouTube accounts for 20–35% of Internet traffic, with 35 hours of videos uploaded every minute and more than 700 billion playbacks in 2010. Hulu had more than 30 million unique viewers in the United States in November 2011 [7]. Netflix currently has 20 million subscribers in the United States, and they consume 30% of North American Internet traffic during peak time [9].

AHS is a new paradigm for video streaming over the Internet currently being used by major content distributors such as YouTube, Netflix, and Hulu. AHS does not depend on specialized video servers. Instead, AHS exploits off-the-shelf web servers. The advantage of this paradigm is that it can easily exploit the widely deployed CDN infrastructure for a scalable video streaming service, and greatly simplifies the firewall and network address translation (NAT) traversal problems. Its weakness is that there is no QoS mechanism on the server side to guarantee video delivery. It relies on the large resource provisioning (i.e., CPU, RAM, storage capacity, storage and network bandwidth, etc.) that is customary with CDNs. This approach achieves simple system design, scalability, and jitter-free video streaming at the cost of large resource overprovisioning.

In the 1990s and early 2000s, Internet-based video streaming for high–bit-rate video was challenging due to technological limitations. In that era, every component of the video server needed to be carefully optimized to meet the volume of clients, the data rate of video, and the real-time guarantees for jitter-free video delivery. For example, real-time scheduling was needed to allocate precious CPU, disk, and network resources for the simultaneous video streams; efficient memory caching was needed to ensure that the disks are not overloaded; a data-placement strategy over multiple disks was needed to evenly distribute the disk load; and an admission-control mechanism was needed to ensure that QoS guarantees can be met without overloading the various system components. However, most of these software solutions have become less important with the remarkable hardware improvements over the past two decades. There is a 100× speedup in CPU speeds; RAM capacity has increased

1 An earlier version of this paper appeared in ACM Multimedia 2013 [43]. This work was supported in part by an unrestricted gift from VMware to the Georgia Tech Foundation in support of this research, and by a research award (CNS-1218520) from the National Science Foundation.

by 1,000×; HDD capacity has grown by 10,000×; and network bandwidth has improved 10,000×. The only thing that has been stagnant is the access latency of the HDD. The access latency has **only** improved four times over the last two decades! [5]

HDDs are currently the primary storage devices that store and serve a large video library. A video service provider should deploy an increasing number of disk arrays to get more storage bandwidth proportional to the rising number of viewers and higher video bit rate (i.e., better picture quality). The amount of user-generated content and the corresponding number of viewers are increasing explosively on the web. In addition, the AHS approach requires storage-bandwidth overprovisioning for reliable on-demand video streaming services; therefore, both the investment cost and the operating cost of storage for a large-scale service is significant. For these reasons, the storage component of the video streaming system needs researchers' attention again for more throughput, lower power consumption, and lower cooling costs.

SSD is a (relatively) new storage technology that comprises semiconductor memory chips (e.g., DRAM, flash memory, phase-change memory) to store and retrieve data rather than using the traditional spinning platters, motor, and moving heads found in conventional magnetic disks. Among various types of SSDs, NAND flash–based SSDs currently have the maximum penetration into modern computer systems. A NAND flash memory SSD can provide higher storage bandwidth than HDDs for a given cost, consume a fraction of the power of HDDs (2 watts of commodity NAND flash SSDs versus 12 watts of commodity HDDs in the active state), and dramatically reduce cooling needs.

Though flash-based SSDs are attractive as an alternative to HDDs for video storage for all the above reasons, the cost per gigabyte for SSD is still significantly higher than for HDDs. Moreover, despite the increasing affordability of SSDs, the ratio of capacity costs of SSD to HDD is expected to remain fairly constant in the future because the bit density of HDDs is also continuously improving. Therefore, it is not cost-effective to replace HDDs entirely with SSDs for permanent video storage. A viable architecture is to use the flash-based SSDs as an intermediate level between RAM and HDDs for caching hot contents, which is known as a multitiered storage architecture.

Multitiered storage with NAND flash memory SSDs has the potential to solve the problems of a traditional two-tier storage system. Flashcache [4] and ZFS [12] are two state-of-the-art solutions that serve as good examples. Both systems are designed to provide higher **average** I/O throughput primarily for general-purpose enterprise applications such as database.

In this paper, we experimentally verify the performance of two state-of-the-art multitiered storage systems—Flashcache and ZFS—to serve as AHS servers. Our experimental results are surprising, because neither of the two systems met our performance expectations. We investigate and report on the limitations of the two systems for adaptive HTTP streaming.[2] To overcome the limitations of the state-of-the-art multitiered storage systems for AHS, we propose *FlashStream*, a multitiered storage architecture incorporating flash memory SSDs that caters to the needs of AHS. The unique contributions of our work are the following:

- We measure performance of two state-of-the-art multitiered storage systems, Flashcache [4] and ZFS [12]. By running an Apache [1] web server on these two systems, we evaluate the storage performance of the systems for AHS workload.
- The performance of both of these systems is surprisingly disappointing. We undertake an in-depth analysis to understand the reasons for the poor performance of both systems for the AHS workload.
- Armed with the knowledge of the sources of poor performance of these state-of-the-art systems, we propose FlashStream, a multitiered storage system that is specially designed for adaptive HTTP streaming workload. We compare the performance and energy efficiency of FlashStream not only with Flashcache and ZFS but also with the traditional two-tier storage architecture.

We built FlashStream as a web server based on the open source Pion Network Library (pion-net) [11]. FlashStream directly manipulates flash memory SSD as a raw device. In addition, FlashStream bypasses the operating system's page cache and manages the DRAM for caching segment files on its own.

The rest of the paper is organized as follows. Section 2 presents the background covering AHS and storage devices. We identify reasons why HDDs might not be an ideal choice for AHS in Section 3. Section 4 measures the performance of the state-of-the-art multitiered storage systems Flashcache and ZFS for AHS workload. In Section 5, we describe our FlashStream architecture for AHS using flash memory SSDs and present extensive experimental results comparing FlashStream with other systems. Related work is presented in Section 6, and the final section presents our concluding remarks.

2. Background

2.1 Adaptive HTTP Streaming Paradigm

AHS is a new video streaming paradigm on the Internet. Rather than rely on the dedicated video servers of yesteryear, AHS exploits off-the-shelf web servers for video streaming. Figure 1 illustrates the mechanism of AHS. A *video object* can be in two different forms: a single large file or multiple small *segment* files that have the same play-out duration, typically a few seconds long. Further, there can be multiple versions of the same video object, supporting different bit rates and different quality levels. Web servers on the Internet that constitute a CDN store these multiple versions of videos. A player (i.e., a client) can then request different segments at different bit rates depending on the state of the underlying network. If the video object is in the form of a single file, the player requests a segment using the byte ranges protocol of HTTP/1.1 [6]. When the video object is in the form of multiple small segment files, the player requests a segment-file download. Notice that it is the player

2 We reported on the performance limitations of the state-of-the-art multitiered storage systems for video streaming in ACM NOSSDAV 2012 [42], and an earlier version of the FlashStream architecture appeared in ACM Multimedia 2013 [43].

that decides the bit rate to request for any segment. The server treats requests for segments of video files similarly to any other normal web request and does not do anything special. This greatly simplifies the server design and enables the use of existing CDN systems without modification.

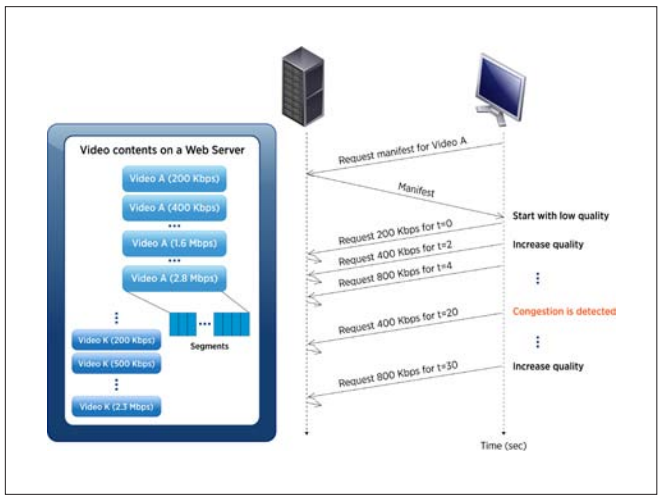


Figure 1. Mechanism of AHS. A video object is divided into segments, and there are multiple versions of the same video object, supporting different bit rates and different quality levels. Web servers on the Internet that constitute a CDN store these multiple versions of videos. A video player (i.e., a client) can request different segments at different bit rates depending on the state of the underlying network via HTTP.

In addition to readily exploiting the widely deployed CDN infrastructure for scalable video streaming, AHS can greatly simplify the firewall and NAT traversal problems. Its weakness is that there is no QoS mechanism on the server side to guarantee video delivery. It relies on the large resource provisioning that is customary with CDNs. This approach achieves simplicity in system design, scalability, and jitter-free video streaming at the cost of large resource provisioning. Netflix is currently operating its service following the paradigm using various CDNs [14]. Moreover, major software companies recently released their own products adopting the AHS paradigm. Microsoft has Smooth Streaming, Apple makes HTTP Live Streaming (HLS), and Adobe has HTTP Dynamic Streaming (HDS). Dynamic Adaptive Streaming over HTTP (DASH) is a standard developed under MPEG, and it became an international standard in November 2011 [8] [48].

2.2 Storage Devices

HDDs have dominated the storage scene in computer systems for several decades by virtue of continuously increasing storage density and driving down the cost per byte of storage. However, new storage technologies such as flash memory and phase-change memory are rising competitors threatening HDDs. SSD is a newer storage device that comprises semiconductor memory chips (e.g., DRAM, Flash memory, phase-change memory) to store and retrieve data rather than using the traditional spinning platters, motor, and moving heads found in conventional magnetic disks. The term *solid-state* means that there are no moving parts in accessing data on the drive. Among various types of SSDs, NAND flash memory SSD nowadays is rapidly penetrating into modern computer systems. NAND flash memory densities have been doubling since 1996, consistently with Hwang's flash memory growth model [26]. Figure 2 shows that the cost trend of flash memory conforms to his estimation [35]. Flash memory was originally designed to be used for mobile devices that require low energy consumption, low heat generation, and shock resistance. However, the sharply dropping cost per gigabyte is what has brought NAND flash memory SSDs to the forefront in recent years. Flash-based storage devices are now considered to have tremendous potential as a new storage medium for enterprise servers [25].

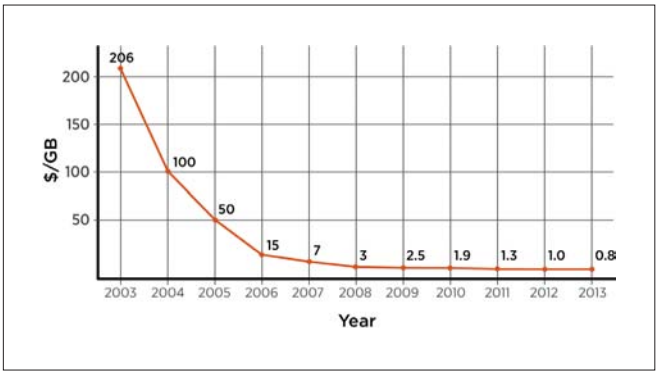
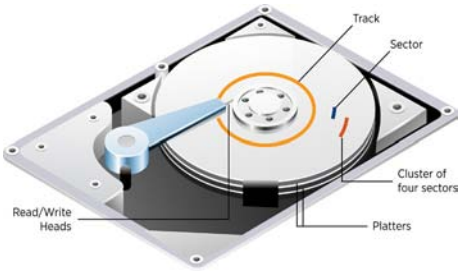


Figure 2. Flash Memory $/GB Trend

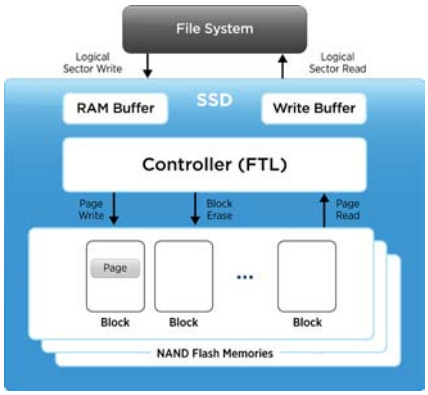
In this section, we will explain the internal architecture and characteristics of the two popular storage devices (i.e., HDDs and flash memory SSDs).

2.2.1 HDD

A HDD consists of one or more rapidly rotating discs (i.e., platters) coated with magnetic material, magnetic heads arranged to read and write data on the disk surface, and a moving arm that moves the heads to a target position. Figure 3(a) depicts the internal architecture of a HDD. Since 1956, when HDDs were introduced by IBM, they have been the dominant secondary storage device. They have maintained this position through technological advances in capacity, reliability, and speed. Access latency of a HDD consists of seek latency and rotational latency. To access data on the surface of the platters, an HDD first moves a head arm to the location of the data. The head arm is a mechanical part and thus it has been challenging to make the seek latency faster. Faster access latency of an HDD is achieved by rotating platters faster, but it increases power consumption, heat generation, and vibration that can jeopardize the reliability of HDDs. For such reasons, HDD access latency has improved **only** four times while HDD capacity has grown by 10,000 times over the past two decades [5]. Average access latency of the latest HDDs is 3–8ms, and energy consumption is 10–15 watts. The competitive cost per unit storage of HDDs has made them the dominant storage device, but new competitors such as flash memory that have much faster speed, lower energy consumption, and lower heat generation, are threatening the HDD's position.



(a) A HDD consists of spinning platters, a moving arm, and magnetic heads to read and write data.



(b) SSD, FTL, and NAND flash memory. FTL emulates sector read and write functionality of a hard disk, enabling conventional disk file systems to be implemented on NAND flash memory.

Figure 3. Architecture of a HDD and a Flash Memory SSD

2.2.2 NAND Flash Memory SSD

Flash memories, including NAND and NOR types, have a common physical restriction: They must be erased before being written to [15]. In flash memory, the number of electrical charges in a transistor is represented by 1 or 0. The charges can be moved both into a transistor by a write operation and out by an erase operation. By design, the erase operation, which sets a storage cell to 1, works on more storage cells at a time than the write operation. Thus, flash memory can be written or read a single page at a time, but it has to be erased in an *erase block* unit. An erase block consists of a certain number of pages. In NAND flash memory, a page is similar to a HDD sector, and its size is usually 2–4 KBytes, whereas an erase block is typically 128 pages or more. Unless otherwise mentioned, flash memory refers to NAND flash memory in this paper.

Flash memory also suffers from a limitation on the maximum number of erase operations possible for each erase block. The insulation layer that prevents electrical charges from dispersing can be damaged after a certain number of erase operations. For SLC NAND flash memory, the expected number of erasures per block is 100,000 but is only 10,000 for two-bit MLC NAND flash memory.

A SSD is simply a set of flash memory chips packaged together with additional circuitry and a special piece of software called the Flash Translation Layer (FTL) [27] [28] [38]. The additional circuitry can include a RAM buffer for storing metadata associated with the internal organization of the SSD and a write buffer for optimizing the write performance of the SSD. The FTL provides an external logical interface to the file system. A sector is the unit of logical access to the flash memory provided by this interface. A page inside the flash memory can contain several such logical sectors. The FTL maps this logical sector to physical locations within individual pages [15]. This interface allows FTL to emulate a HDD as far as the file system is concerned (Figure 3(b)).

To avoid erasing and rewriting an entire block for every page modification, FTL writes data out of place, remapping the logical page to a new physical location and marking the old page invalid. This requires maintaining some quantity of free blocks into which new pages can be written. These free blocks are maintained by erasing previously written blocks to allow space occupied by invalid pages to be made available for new writes. This process is called *garbage collection*. FTL tries to run this process in the background as much as possible while the foreground I/O requests are idle, but it is not guaranteed, especially when a new clean block is needed instantly to write a new page. Due to random writes emanating from the upper layers of the operating system, a block can have valid pages and invalid pages. Therefore, when the garbage collector reclaims a block, the valid pages of the block need to be copied to another block. Thus, an external write can generate some additional unrelated writes internal to the device, a phenomenon referred to as *write amplification*. Complicated FTL mapping algorithms that require more resources have been proposed to get better random write performance [38]. However, due to the increased resource usage of these approaches, they are used usually for high-end SSDs. Even though high-end SSDs using fine-grained FTL mapping schemes can provide good random write performances, the effect of background garbage collection will be a problem, considering the soft real-time requirements of video-on-demand (VoD) server systems.

The advantages of flash-based SSDs are fast random read (0.05–0.1ms), low power consumption (1–2 watts per drive), and low heat generation due to the absence of the mechanical components. On the other hand, its high cost per gigabyte compared to magnetic disks, poor small random write performance, and limited lifetime are major concerns.

Narayanan et al. [37] have analyzed the efficacy of using flash-based SSDs for enterprise-class storage via simulation. In addition to using flash exclusively as the permanent storage, they also study a tiered model wherein the SSD is in between the RAM and the disks. In their study they use enterprise-class SSDs ($23/GB). Their conclusion is that SSD is not cost-effective for most of the workloads they studied unless the cost per gigabyte for SSD drops by a factor of 3 to 3,000. Their results are predicated on the assumption that SSD is used as a transparent block-level device with no change to the software stack (i.e., application, file system, or storage system layers). The results we report in this paper offer an interesting counterpoint to their conclusion. In particular, we show that the use of inexpensive commodity SSDs as a buffer cache is a cost-effective alternative for structuring a VoD server, as opposed to increasing either the disk bandwidth or RAM capacity to meet a given QoS constraint.

3. Storage Performance for AHS

For a VoD system, HDDs have long been used to store and serve video data because of their large capacity per dollar and high throughput for accessing video files that are typically large in size. However, this premise does not hold anymore for AHS systems. In this section, we will explain our reasoning behind such a claim.

3.1 Segment Size in AHS

As we explained in Section 2.1, a segment (i.e., a part of a video object) is the access granularity in AHS. The size of such segments is highly variable for the following reasons. In AHS systems, every video segment has the same play-out duration. Though each segment has the same play-out time, the size of each segment is different because the video is encoded with a variable bit rate (VBR). In addition, the segment length can be short or long depending on a video publisher's decision. For example, Microsoft's Smooth Streaming uses 2-second segment length by default, and Apple's HTTP Live Streaming uses 10-second segment length by default [19]. The segment size is proportional to the length of a segment. Moreover, a single video object is encoded in multiple bit rates in the AHS systems. Because the segment size is proportional to the bit rate, lower–bit-rate segments would have a smaller size, whereas higher–bit-rate segments would have a larger size. Therefore, the size of segments in the AHS systems can span from a few kilobytes to a few megabytes. DASH [8] is an ISO/IEC MPEG standard of the AHS paradigm. Table 1 shows DASH dataset that is publicly available for benchmarking.

NAME	BIT RATES (KBPS)	LENGTH	GENRE
Big Buck Bunny	50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000, 8000	09:46	Animation
Elephants Dream	50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000, 8000	10:54	Animation
Red Bull Playstreets	100, 150, 200, 250, 300, 400, 500, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000	97:28	Sport
The Swiss Account	100, 150, 200, 250, 300, 400, 500, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000	57:34	Sport
Valkaama	50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 900, 1100, 1400, 1700, 2000, 2500, 3000, 4000, 5000, 6000	93:05	Movie
Of Forest and Men	50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 900, 1100, 1400, 1700, 2000, 2500, 3000, 4000, 5000, 6000	10:53	Movie

Table 1. DASH Dataset

We measured the size of segments of DASH datasets in two different segment lengths. Figure 4 shows the cumulative distribution function (CDF) of segment sizes for 2-second-long segments. For 2-second-long segments, 50% are less than 184KB in size, and the standard deviation is 386KB. Figure 5 shows CDF of segment sizes for 10-second-long segments. For 10-second-long segments, 50% are less than 878KB in size, and the standard deviation is 1865KB. These graphs tell us that segment sizes can be very small and diverse in AHS. In addition, AHS systems replicate video segments to a number of CDN cache servers for effective load balancing. A client's request for a segment is directed to a cache server holding the segment via a request-routing technique. There are many different request-routing techniques, such as DNS routing, HTML rewriting [18], and anycasting [39]. For this reason, there is no guarantee that a client who downloaded a segment i of a video will download the next segment $i+1$ from the same server. The next segment request might be directed to a different cache server that holds a replica. The small and variable segment sizes and the load-balancing mechanism of AHS systems make a storage system challenging.

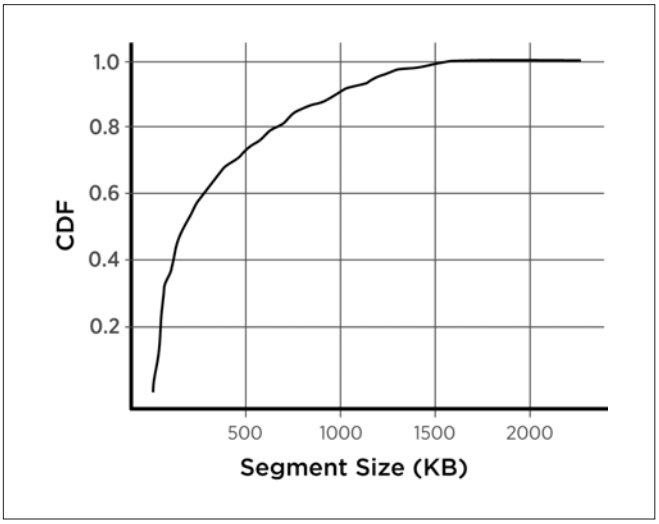


Figure 4. CDF of Segment Sizes of DASH Dataset for 2-Second-Long Segments. 50% of segments are less than 184KB, and the standard deviation is 386KB.

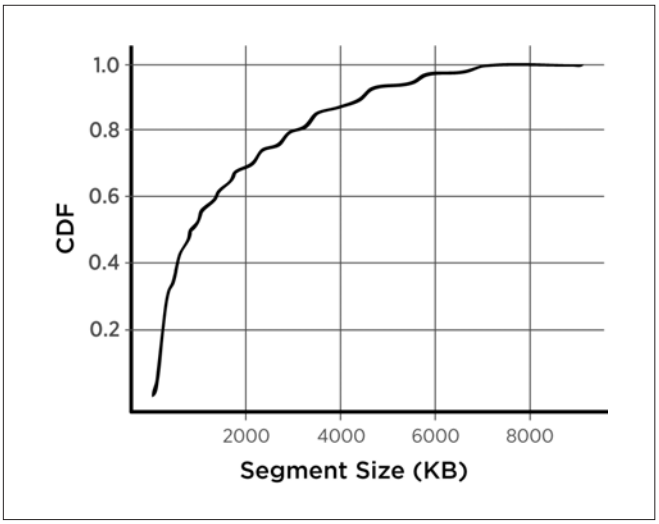


Figure 5. CDF of Segment Sizes of DASH Dataset for 10-Seconc-Long Segments. 50% of segments are less than 878KB, and the standard deviation is 1865KB.

3.2 HDD Performance

A HDD consists of one or more rapidly rotating discs (i.e., platters) coated with magnetic material, magnetic heads arranged to read and write data on the disk surface, and a moving arm that moves the heads to a target position. Access latency of a HDD consists of seek latency and rotational latency. Seek latency is the time taken to move a head arm to the track of a platter where a target sector belongs to. Rotational latency is the time taken to rotate the platter for the head arm to meet the target sector on the track. Between the two, seek latency is usually longer than rotational latency. For this reason, HDDs are very slow when accessing small data randomly. Figure 6 shows an HDD's read throughput in MB/sec against different request sizes. When accessing 4KB of data randomly, the HDD's throughput is 0.4MB/sec, whereas the random access of 64MB of data shows 99MB/sec throughput. In the previous section, we observed that segment-file sizes are very small and diverse in AHS. In addition, a CDN cache server receives random segment requests from clients due to the load-balancing mechanism of AHS systems. In the previous section, we identified that the median segment size of DASH dataset is 184KB when the segment length is 2 seconds. With a 184KB request size, the random read throughput of the HDD is very low (approximately 15MB/sec). Therefore, HDDs are not ideal for streaming video segments in AHS systems.

3.3 Flash Memory SSD Performance

SSD is a (relatively) new storage device that comprises semiconductor memory chips (e.g., DRAM, flash memory, phase-change memory) to store and retrieve data rather than using the traditional spinning platters, motor, and moving heads found in hard disk drives. Among various memory technologies, NAND flash memory is the technology that is widely used for SSDs nowadays. When we say *flash memory SSDs* in this paper, it means NAND Flash memory SSDs. One advantage of flash-based SSDs compared to HDDs is fast random read (0.05–0.1ms). Figure 7 shows a SSD's read throughput in MB/sec against different request sizes. The median segment size of the DASH dataset is 184KB when the segment length is 2 seconds (refer to Section 3.1). With a 184KB request size, the SSD can provide around 210MB/sec throughput for random reads, whereas HDD's random read throughput is approximately 15MB/sec. Therefore, a flash memory SSD is a promising storage device for AHS.

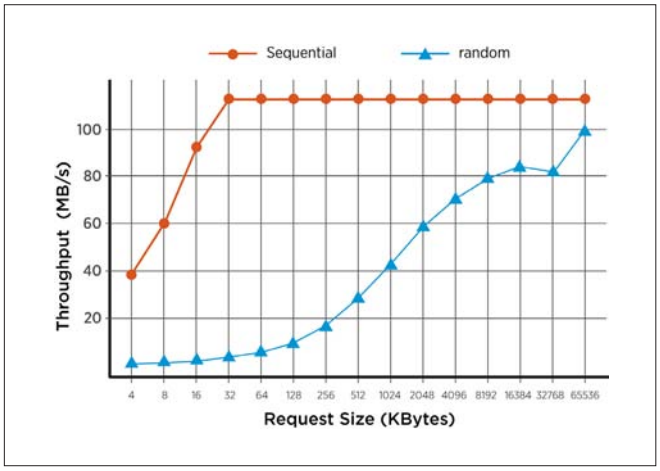


Figure 6. HDD's Read Throughput Against Different Request Sizes. 7200RPM HDD is used. HDDs show poor performance when reading small data randomly.

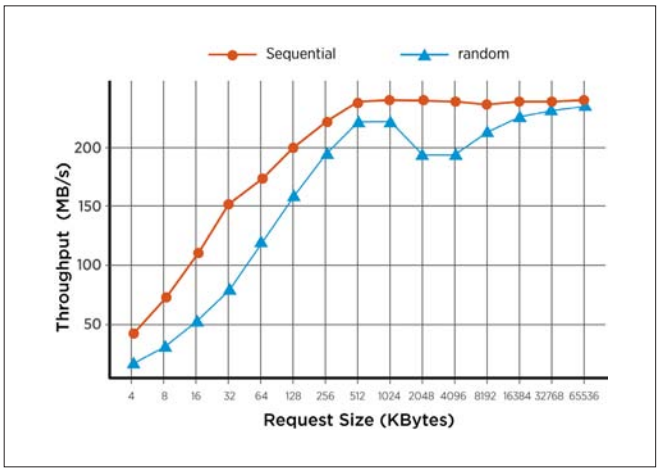


Figure 7. SSD's Read Throughput Against Different Request Sizes. Intel X25-M-G1 is used. SSDs show very good performance for random reads.

3.4 Summary

In AHS systems, a video object is accessed at the granularity of a segment. Such a segment is encoded in multiple bit rates (from low to high bit rate), and the segment length can be short (1 to 2 seconds). Therefore, the size of segments in the AHS systems can span from a few kilobytes to a few megabytes. In addition, the segments are accessed randomly in the CDN cache servers. We identified that the median segment size of a DASH dataset is 184KB when the segment length is 2 seconds. With a 184KB request size, the random read throughput of the HDD is very low (approximately 15MB/sec). In contrast, flash-based SSDs can provide very large random read throughput (approximately 210MB/sec) for streaming such small segments. Therefore, HDD is not necessarily an ideal storage device for streaming videos using the AHS paradigm.

4. Multitiered Storage Systems

A large-scale VoD system requires a number of HDDs both for capacity (to store the video library) and for bandwidth (to serve the video library). While the cost per gigabyte of HDDs has decreased significantly, the cost per bits-per-second of HDDs has not. Moreover, an array of HDDs consumes a lot of power (approximately 5–15 watts per drive) and generates a large amount of heat; therefore, more power is required for cooling a data center hosting a large array of disks. The amount of user-generated content and the corresponding number of viewers are increasing explosively on the web. In addition, the AHS approach requires storage bandwidth overprovisioning for reliable video streaming service; therefore, the cost of storage for a large-scale service is significant.

Flash memory SSDs open up new opportunities for providing a more cost-effective solution for a VoD storage system. The advantages of flash-based SSDs are fast random read (0.05–0.1ms), low power consumption (1–2 watts per drive), and low heat generation due to

the absence of the mechanical components. Though flash-based SSDs are attractive as an alternative to HDDs for video storage for all the above reasons, the cost per gigabyte for SSD is still significantly higher than for HDDs. Moreover, despite the increasing affordability of SSDs, the ratio of capacity costs of SSD to HDD is expected to remain fairly constant in the future because the bit density of HDDs is still continuously improving. Therefore, a viable architecture is to use flash-based SSDs as an intermediate level between RAM and HDDs for caching hot contents. Such a storage architecture is called a *multitiered storage system*.

Flashcache [4] and ZFS [12] are state-of-the-art commercial multitiered storage systems incorporating flash memory SSDs. Flashcache, developed by Facebook, is a write-back persistent block cache designed to accelerate reads and writes from slow storage such as HDDs by caching data in faster storage such as SSDs. ZFS [12] is a file system that has an intermediate layer to serve as a read cache between the RAM and the HDDs, called L2ARC. Storage devices that are faster than HDDs are used for L2ARC devices. Though not a requirement, flash memory SSDs can be used as L2ARC devices. We have measured the two state-of-the-art flash-based multitiered storage systems, and both have shown quite disappointing performance for AHS workloads. The poor performance of Flashcache and ZFS is very surprising because both systems are designed with flash memory SSDs in mind. In this section, we analyze the reasons behind the poor performance of Flashcache and ZFS. Based on the lessons learned from the analysis, we provide design guidelines for a multitiered storage system for AHS. Unless otherwise mentioned, a *cache* refers to a flash memory SSD rather than RAM in this section.

4.1 Measurement

We measure the performance of three different storage configurations of an HTTP streaming server: HDDs only, Flashcache, and ZFS. Apache [1] is used for a web server. We implement a workload generator that emulates a large number of concurrent AHS clients. To measure the storage subsystem performance exactly and to avoid network subsystem effects, the Apache web server and the workload generator run on the **same** machine, communicating via a loop-back interface. The machine has a Xeon 2.26GHz Quad core processor with 4GB RAM, and Linux kernel 2.6.32 is installed on it. We use two 7200RPM HDDs striped per Linux's software RAID-0 configuration. Intel X25-M G1 and OCZ Core V2 are flash memory SSDs that are used in experiments, but we will show results with only Intel X25-M G1 because results with OCZ Core V2 are similar. Unless otherwise noted, cache refers to the flash device used as an intermediate level in the storage hierarchy and not RAM.

4.1.1 Workload

Zipf distribution is generally used in modeling the video-access pattern of a VoD system, and typically a parameter value between 0.2 and 0.4 is chosen for the distribution [36]. We use 0.271 for the parameter. DASH [8] is an ISO/IEC MPEG standard of the AHS paradigm. We use a DASH dataset [31] for our test that is available in the public domain. Among videos in the dataset, we use *Valkaama* for a test video sequence in this study. The video object is segmented into 10-second-long segments, the total length of the video is about 78 minutes, the size is 1.06GB, and the average bit rate is 2Mbps. The video object is composed of 466 segment files. Though each segment has the same play-out time, the size of each segment is different because the video is encoded with a VBR. We copied the video and make 300 distinct video objects. Figure 8 shows the zipf distribution of the 300 videos.

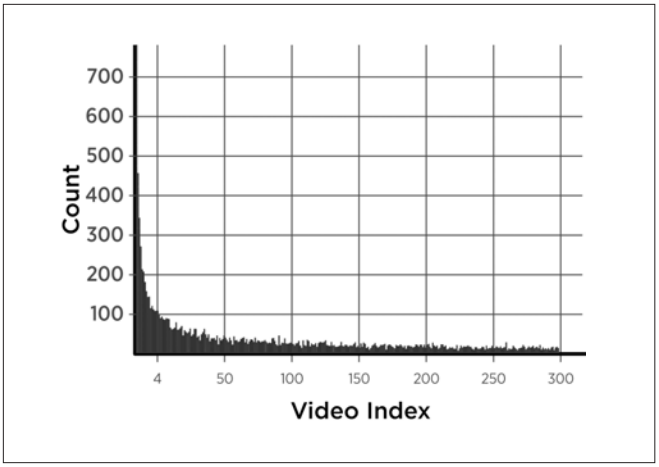


Figure 8. Distribution of Access Frequency of 300 Videos. The zipf parameter is 0.271. Top 60 (20%) popular videos account for 57.6% of total requests.

Every *t* seconds (which is the inverse of the request rate), the workload generator selects a video object according to the zipf distribution. Next, it chooses a segment of the video object according to the uniform distribution; each segment of the video object has the same probability. The reason for using a uniform distribution is as follows. A large-scale HTTP video streaming service such as Netflix relies on the CDN infrastructure that widely deploys web cache servers near the edge networks. For effective load balancing, a video segment (or object) is replicated to a number of web cache servers, and a client's request for the segment is directed to a web server holding the segment via request-routing techniques such as DNS routing, HTML rewriting [18], or anycasting [39]. For this reason, there is no guarantee that a client who downloaded a segment i of a video will download a next segment $i+1$ from the same server. The next segment can be served by other web servers that hold a replica. Therefore, it is reasonable to assume a uniform distribution of segment requests to any given web server.

The workload generator sends an HTTP request for the chosen segment to the web server. When the segment is not downloaded to the client within the segment's play-out time (i.e., 10 seconds for our test video), the client counts it as a segment deadline miss. We choose the 10-second segment size because it is the default size used by Apple HTTP Live Streaming [19]. We measure the *segment miss ratio* against different request rates. Segment miss ratio is defined as the ratio of the number of segment deadline misses to the total number of segment requests for a given request rate. Therefore, the *requests per second* is a control parameter, and the segment miss ratio is the measured figure of merit of the storage subsystem.

4.1.2 Measurement Results

As a base configuration for comparison, we use two 7200RPM HDDs striped per Linux's software RAID-0 configuration. Software RAID is implemented in Linux using the device mapper, which is a Linux storage-stack infrastructure. The ext4 file system [3] is installed on

these RAID-0 disks. We measure the segment miss ratio with different request rates, and each measurement is run for an hour. Figure 9 shows that for this configuration the system could serve up to 19 requests per second when the required QoS is 0% segment miss ratio. Because the observed average CPU utilization during the measurements was below 10%, we can conclude that storage is the bottleneck beyond 19 requests per second for this configuration.

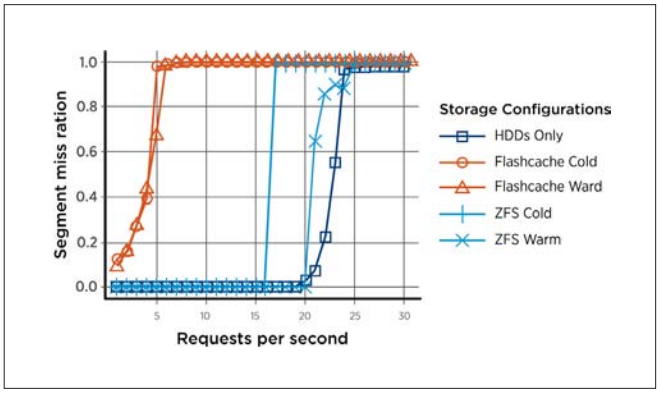


Figure 9. Segment Miss Ratio as a Function of Request Base.

Flashcache [4], developed by Facebook, is a write-back persistent block cache designed to accelerate reads and writes from slow storage such as HDDs by caching data in faster storage such as SSDs. Flashcache is a block-device-level solution; therefore, it is very general and can be utilized at different levels of the software stack (e.g., file systems, applications). We create a single logical block device by having Flashcache use the two 7200RPM HDDs striped per RAID-0 configuration and the Intel X25-M SSD. Out of the total 80GB available in the SSD, 60GB are used as Flashcache for the experiment. We measure the performance in two different cache states: a cold cache and a warm cache. For the cold cache, we flush the cache at the beginning of each measurement. For the warm cache, we run the workload generator until the cache filling rate falls below 100KB/sec. We fill the cache up to 60% (i.e., 36GB) by running the workload generator with 1 request per second for 12 hours. Each measurement is run for an hour. Flashcache shows surprisingly poor performance. Figure 9 shows that Flashcache could not serve even 1 request per second when the required segment miss ratio is 0%. The Flashcache performance is the same whether the cache is cold or warmed up.

ZFS [12] is a file system that has an intermediate layer to serve as a read cache between the RAM and the HDDs, called L2ARC. Storage devices that are faster than HDDs are used for L2ARC devices. Though not a requirement, flash memory SSDs can be used as L2ARC devices. ZFS was originally introduced and implemented in the Solaris operating system, but it has been ported to other operating systems such as FreeBSD and Linux. We use the version of ZFS that is ported to Linux for this measurement. ZFS creates a single storage pool using the two 7200RPM HDDs in RAID-0 configuration together with the Intel X25-M SSD to serve as the intermediate read cache layer (i.e., L2ARC). As in the Flashcache experiment, we use 60GB of the SSD capacity for the L2ARC cache and measure the performance in two different cache states: a cold cache and a warm cache. For the cold cache, we flush the cache at the beginning of each measurement. For the warm cache, we run the workload generator until the cache is full. Each measurement is run for an hour. Figure 9 shows that ZFS with the cold cache could serve up to 16 requests per second when the required segment miss ratio is 0%. When the cache is warmed up, ZFS could serve up to 20 requests per second. ZFS shows a lot better performance than Flashcache, but it is still worse than the base configuration (HDDs-only) when the cache is cold and only slightly better after the cache is warmed up.

4.2 Analysis

The poor performance of Flashcache and ZFS is very surprising because both ZFS and Flashcache are designed with flash memory SSDs in mind. In this section, we investigate the reasons for this result.

4.2.1 Flashcache

Flashcache organizes flash memory as a set-associative cache. The block size, set associativity, and cache size are configurable parameters, specified at cache creation time. The default block size is 8 sectors (i.e., 4KB), and the default set associativity is 512 (i.e., a given disk block can be in one of 512 members of a given set) The replacement policy is either First In First Out (FIFO) or Least Recently Used (LRU) within a set, and FIFO is the default. We used default values for all the parameters in our Flashcache measurements.

In what follows, *dbn* refers to *disk block number*, the logical device block number in sectors. To compute the target set for a given dbn:

$$target\ set = \left(\frac{dbn}{block\,size \times associativity} \right) mod\ (number\ of\ sets)$$

After we have the target set, Flashcache does a sequential search of the set (linear probing) to find the desired disk block. Note that a sequential range of disk blocks will all map onto a given set. Disk blocks that have a dbn difference greater than the associativity will map onto different sets.

When Flashcache gets a read request, it calculates the target set from the dbn of the request. Then, it probes the cache set to find the requested block. If it finds the block in the cache, it returns the block. Otherwise, Flashcache reads the block from the disk, copies the block into the appropriate cache line, and returns the block. A block request that misses in the cache will first write the block into the cache before returning it to the application. Thus the cache write operation is in the critical path of satisfying an application-layer block request that misses in the cache. Therefore, random reads on disks that miss the cache will generate random writes to the cache, and the read could be delayed and miss its deadline if the cache write operation takes a long time to complete.

Figure 10[3] depicts the read and write access patterns on the SSD for Flashcache in two different cache states for an hour period with 1 request per second. This graph is plotted based on the trace data

3 The purpose of Figure 10 is to show that both read and write access patterns for cold and warm cache settings are completely random, as is evident from the distribution of the data points in the scatter plot shown in the figure.

collected using blktrace [2], which comes with the Linux 2.6 kernel, to trace the I/O activities at the block device level. To remind the reader, read requests to the flash come from the client requesting video segments that are already cached in the flash; the write requests come from the need to copy blocks from the disk to the flash for requests that miss the flash. The upshot is that both the read and write access patterns are random, and the median write request size is 116KB (see Figure 11). The request size is defined by the I/O request size sent by the block device driver to the physical block device, which we can determine using blktrace. Flash memory shows the best write performance when the request size is a multiple of the erase block size, which is 32MB for the SSD. A write request size of 116KB is much smaller than the optimal write request size (i.e., 32MB). The effect of these small random writes is disastrous on the flash memory performance. In particular, because flash memory does not support in-place writes, requires erase-before-write, and also requires that the erase size be larger than the page size, the small and random write operations will consume fresh pages in a clean block. Ultimately this will lead to the situation in which all available clean blocks are used up, thus kick-starting the garbage-collection process. Looking at the top chart in Figure 12, which shows the SSD utilization, we can see the utilization peaking up to 100% periodically (around every 140 seconds). It is highly likely that this is due to the garbage-collection process. Overall, Flashcache generates a very inefficient access pattern for the flash memory SSD.

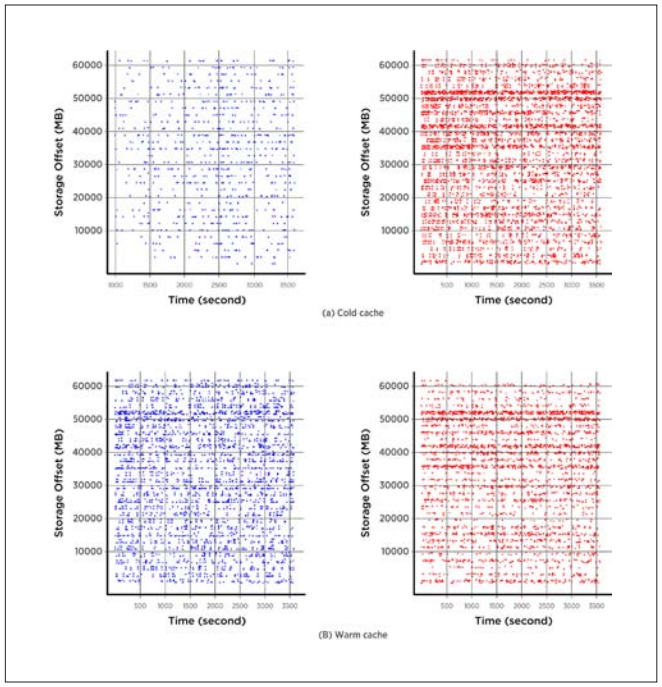


Figure 10. Flashcache's Read (Blue) and Write (Red) Access Pattern on the Cache During 1 Hour with 1 Request per Second. The x-axis is time and the y-axis is the storage offset. Both read and write access patterns are severely random.

The read hit ratio of Flashcache with a cold cache (Figure 10(a)) is 0.7%, whereas the read hit ratio of Flashcache with warm cache (Figure 10(b)) is 25.7%. However, as shown in Figure 9, Flashcache shows absolutely no difference in performance as measured by the application-level segment miss ratio. This can be explained as follows. From the scatter plot for write requests shown in Figure 10(b), we

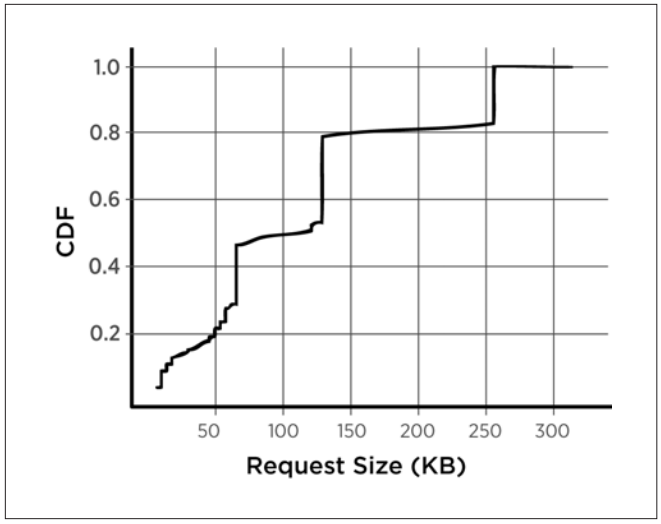


Figure 11. Cumulative Distribution Function (CDF) Plotted Against the Write Request Sizes Sent to Flashcache over a Period of 1 Hour. The median write request size is 116KB.

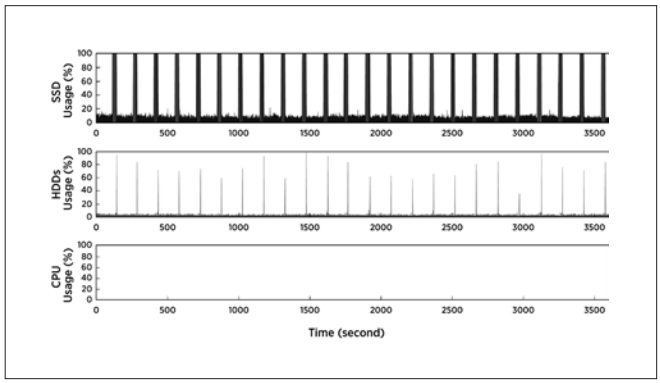


Figure 12. Flashcache Resource Utilization During 1 Hour with 1 Request per Second and Cold Cache.

notice that there is a significant volume of write requests to the cache even when the cache is warm. Thus, the application-generated reads and system-generated writes (a consequence of miss handling) are competing for resources on the SSD device (RAM buffer and CPU). Further, the device processes the requests in order. Thus, if the reads are queued up behind writes, they are likely to miss their deadline even though they hit in the cache, because random writes take a long time to complete. This is the most likely reason for the poor application-level segment miss ratio observed even with a warm cache in Figure 9 for Flashcache.

The reasons for the high segment miss ratio for Flashcache while dealing with the AHS workload can be summarized as follows:

- Upon a cache miss, the block read from the disk has to be first written to the cache before being served to the application. The ensuing small and random write pattern results in long latencies.
- Because Flashcache does not give priority to reads over writes to the cache, reads that would be hits in the cache get queued up behind ongoing writes.

Serving the requested segments should be the top priority for a video streaming system. Not respecting this criterion is ultimately the failing of Flashcache for this workload.

4.2.2 ZFS

ZFS is smarter than Flashcache in handling flash memory. By analyzing the source code for ZFS L2ARC, we have determined that ZFS basically uses L2ARC as a FIFO buffer. By definition, the blocks written into the cache are clean (i.e., they are just copies of the disk blocks) and therefore they never have to be written back to the disk. ZFS maintains a *write pointer* to the FIFO buffer that L2ARC represents. When the write pointer reaches the end of the FIFO buffer it simply wraps around to the beginning, overwriting the existing blocks in the cache. In other words, ZFS L2ARC converts the random writes to sequential writes. This is evident from Figure 13(b), where it can be seen that ZFS L2ARC sequentially fills up the cache over time.

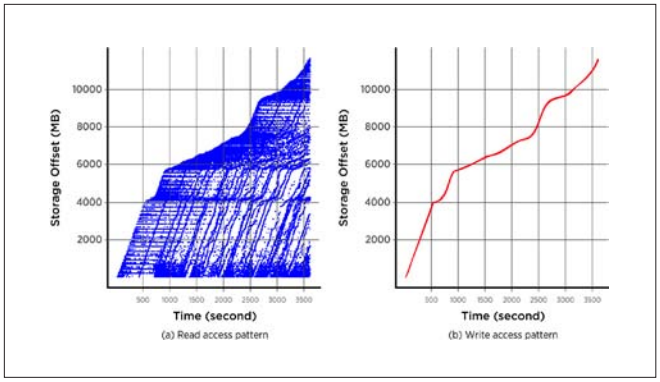


Figure 13. ZFS's Read (Blue) and Write (Red) Access Pattern on the Cache During 1 Hour with 16 Requests per Second and Cold Cache. The write access pattern is sequential, whereas the read access pattern is random. The median write request size is 128KB.

We measured the median write size to be 128KB from the block traces. Interestingly, we ascertained by executing a microbenchmark for the SSD that its sequential write throughput peaks at 128KB. Figure 14 also corroborates our analysis. The top graph in Figure 14 is the activity observed on the SSD. It can be seen that this activity is very low, indicating that the SSD performance is not the bottleneck. Thus, overall ZFS L2ARC seems to be optimized in handling the flash memory SSDs. That begs the question as to why ZFS performs much worse than the base configuration (HDDs-only), as shown in Figure 9.

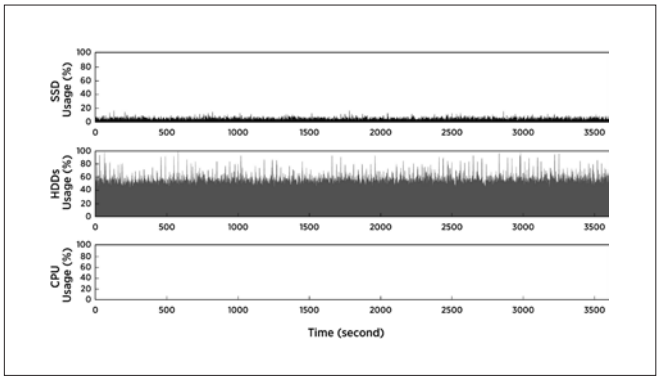


Figure 14. ZFS's Resource Utilization During 1 Hour with 20 Requests per Second and Warm Cache.

The answer is simply that L2ARC is being used as a FIFO buffer. In other words, the replacement policy for the L2ARC cache is FIFO. As we noted, treating L2ARC as a FIFO buffer is great for write throughput considering the nature of the flash device. However, this leads to a very poor hit ratio. Unfortunately, it is not possible to empirically verify this hypothesis, because the Linux port of ZFS does not provide statistics such as L2ARC hit ratio. Therefore, we approximate the hit ratio from the amount of read size requested from the cache and from the disks, which can be obtained by analyzing the block trace. We use following formula to approximate the hit ratio:

$$Hit\ Ratio = \frac{Amount\ of\ read\ on\ SSD}{Amount\ of\ read\ on\ SSD + Amount\ of\ read\ on\ disks}$$

Using this approximation, L2ARC hit ratio is 4.2% and 11.1% for the cold cache and the warm cache, respectively. The low hit ratio is reflected in the low SSD utilization shown in Figure 14.

We have also applied the above formula to Flashcache, and the hit ratio is 0.67% and 23.6% for the cold cache and the warm cache respectively. Compare these numbers to the statistics that Flashcache provides. They are 0.7% and 25.7% for the cold cache and the warm cache respectively (refer to Section 4.2.1). Therefore, we can say that our approximation for the hit ratio derived from block traces is close to reality.

4.3 Discussion

By studying and analyzing the performance of state-of-the-art multitiered storage systems for AHS, we learned a number of lessons. We summarize these lessons in the form of recommendations for constructing a cost-conscious high-performance multitiered storage system for AHS.

Recommendation 1: No small random writes – Small random writes are extremely inefficient for flash memory SSDs. There are two possible solutions to this problem. The first solution is a logging approach. This approach transforms the random writes to sequential writes that the flash memory SSDs can handle very efficiently. However, logging necessitates frequent invocation of the garbage collection any time it needs to clean obsolete blocks and make room for new data blocks that need to be written. Clearly, this is detrimental to real-time performance such as timely video delivery, because foreground read operations can be stymied and delayed, resulting in missing application deadlines. Therefore, logging is not an appropriate solution for the small-random-write problem in a real-time system like video streaming. A more preferred solution is to write using a much larger granularity. If write operations are requested in multiples of the flash memory's erase block size, and their offset is aligned with multiples of the erase block size, write amplification will not occur, and flash memory SSDs can handle the writes very efficiently even if the access pattern is completely random.

Recommendation 2: No flash writes on the critical path – Writes to the cache during servicing of a cache miss should not be in the critical path of serving the requested segment to the application. Hardware caches in a processor are routinely designed in this fashion, wherein the missing data (due to load instruction) is supplied to the

processor in parallel with updating the cache. Failing to do this in a video server guarantees that a read request that misses in the cache will incur the additional penalty of write to the cache when the data is brought from the hard disk. ZFS solves this problem using an evict-ahead policy by which a separate thread copies blocks that are supposed to be evicted soon from RAM to the flash [35] [13].

Recommendation 3: Higher priority for reads – Flash reads are more important than flash writes because the former need to be served before their deadline, whereas the latter are not time-critical. Therefore, flash reads should have a higher priority than flash writes when they compete for resources. Both Flashcache and ZFS do not consider this point.

4.4 Summary

Due to the cost and size advantage of flash memory compared to DRAM, a multitiered storage hierarchy—wherein a flash-based SSD serves as an intermediate-level cache—appears to be an attractive strategy for constructing a cost-efficient high-performance video streaming server. However, we found through extensive performance studies that two state-of-the-art multitiered storage systems exhibited disappointingly poor performance for AHS. We performed careful analysis to uncover the sources of poor performance in both of these systems. Based on our analysis, we have made recommendations for constructing a multitiered storage system for AHS that avoids the pitfalls in designing such a system.

5. FlashStream: A Multitiered Storage Architecture for AHS

In the previous section, we learned that the current state of the art in multitiered storage systems such as ZFS and Flashcache, architected for general-purpose enterprise workloads, do not cater to the unique needs of AHS. The lessons learned from Section 4.3 can be summarized as follows:

- Although flash memory SSDs are well-suited to serve the caching needs of videos in an HTTP streaming server, it is of paramount importance to avoid small random writes to the SSDs, due to the unique performance characteristics of flash memory.
- Flash write operations should not be in the critical path of serving missed video segments (brought from the hard disk) to the clients.
- Flash read operations should have higher priority than flash write operations because the former need to be served before their deadline to the clients, whereas the latter are not time-critical.

In this section, we propose FlashStream, a multitiered storage architecture incorporating flash memory SSDs that caters to the needs of AHS. The unique contributions of our work are the following:

- To minimize write amplification, writes to SSDs are always requested at the granularity of an optimal block size and aligned on the optimal block size. In addition, we present a novel reverse-engineering technique to find out the optimal block size for any flash-based SSD. This accomplishes the first design guideline.
- Utilization-aware SSD admission and ring buffer mechanisms control the write request rate to SSDs and give higher priority to read requests. We adopt ZFS's evict-ahead policy to avoid SSD write operations from the critical path of serving missed data. This fulfills the second and third design guidelines.
- Metadata for video segments are embedded in the SSD blocks to quickly restore data in SSD upon power failure.

FlashStream is designed and implemented based on the guidelines presented in Section 4.3. We experimentally evaluated the performance of FlashStream and compared it to both ZFS and Flashcache. We show that FlashStream outperforms both of these systems for the same hardware configuration.

5.1 Optimal Block Size

As we discussed in Section 2.2.2, the reasons for the poor performance of flash memory for small random writes are twofold: (a) A block has to be erased before a page within it can be written, and (b) an erase block consists of several pages, valid pages in the block being erased must be copied to other clean pages before erasing the block (*write amplification*), and page write and block erasure are inherently slow operations. Due to the write-amplification effect, small random writes not only degrade the SSD performance but also reduce the flash memory lifetime.

One way to completely eliminate both write amplification and poor performance due to random page writes is to ensure that write requests to the SSD are always in multiples of the block size of the flash memory that is used in the SSD, and that the writes are aligned on block boundaries. This strategy will also have the additional benefit of fast garbage collection (due to the elimination of write amplification) and longer flash memory lifetime. However, there is a catch.

SSD manufacturers put great effort into fully exploiting all available hardware resources and features to improve performance, and they use different designs [15]. In addition, the internal organization of a SSD and their techniques are proprietary and not readily available. How do we choose the right write granularity in the software architecture of the storage system when we do not know the internal architecture of the SSD? To this end, we define the optimal block size for write requests to the SSD as the minimum write size that makes the random write performance similar to the sequential write performance.

To determine the optimal block size, we do a reverse-engineering experiment on any given flash-based SSD. The idea is to perform random writes with different request sizes until the performance matches that with a sequential workload. To illustrate the technique, we use an OCZ Core V2 flash-based SSD. Figure 15 shows the latency of 640 write operations for sequential and random workloads when the request size is 8MB on this SSD. For the random writes (see Figure 15(c)), we notice frequent high latencies (i.e., 800–1000ms) due to the write-amplification effect. When the request size is 16MB (see Figure 16), the latency distributions for the sequential write and

the random write workloads are very similar. Figures 15(b) and 15(d) show the CDF graphs for the 8MB request size; Figures 16(b) and 16(d) show the CDF graphs for the 16MB request size. Although the CDF graphs are useful for seeing the similarity between two distributions, we need a quantitative measure to determine the similarity.

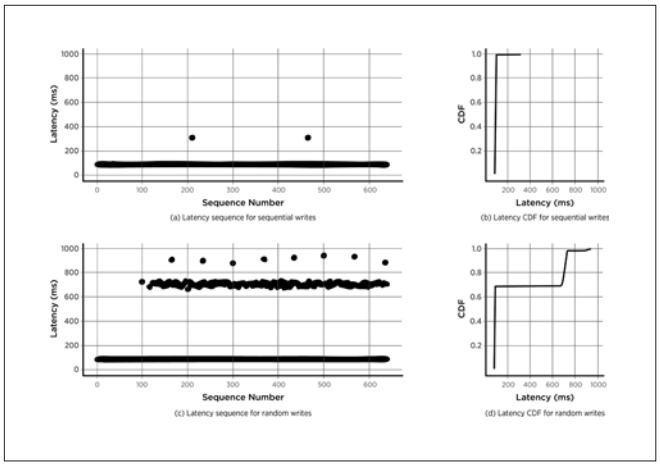


Figure 15. Latency Distribution for Sequential/Random Writes with 8MB Request Size. OCZ Core V2 is used for the measurement.

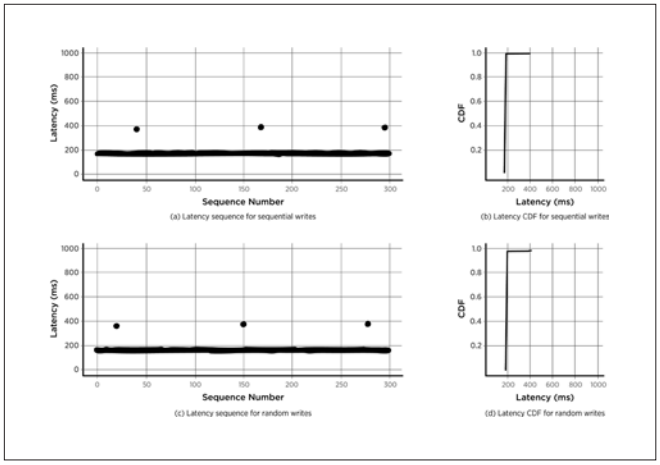


Figure 16. Latency Distribution for Sequential/Random Writes with 16MB Request Size. OCZ Core V2 is used for the measurement.

Kullback-Leibler (KL) divergence [30] is a fundamental equation from information theory that quantifies the proximity of two probability distributions; the smaller the KL value, the more similar the two distributions. In flash memory SSDs, write-operation latency is a random variable and hard to estimate because it is affected by various factors such as mapping status, garbage collection, and wear-leveling. We use the KL divergence value as the metric for comparing the similarity of the latency distributions obtained with the sequential and random write workloads.

Figure 17 shows how the KL divergence value changes with different request sizes for three different SSDs. For all the SSDs, the KL divergence value converges to zero as the request size increases. We let the optimal block size be the request size at which the KL divergence value becomes lower than a threshold that is close to zero. The threshold is a configuration parameter. A lower threshold can give a block size that shows more-similar latency between sequential writes and random writes. We choose 0.1 for the threshold because Figure 17 shows the KL divergence value quickly converges to zero when it becomes lower than 0.1. Therefore, 64MB is the optimal block size for INTEL X25-M G1, 16MB for OCZ Core V2, and 4MB for SAMSUNG SLC MCCOE64G5MPP. The optimal block size is a function of the internal architecture of the SSD (page size, block size, available internal parallelism for access to the flash memory chips, etc.). However, this experimental methodology enables us to reverse-engineer and pick a write size that is most appropriate for a given SSD. After the optimal block size has been determined, FlashStream divides the logical address space of flash memory into identical allocation blocks that are of the same size as the optimal block size determined with the experimental methodology.

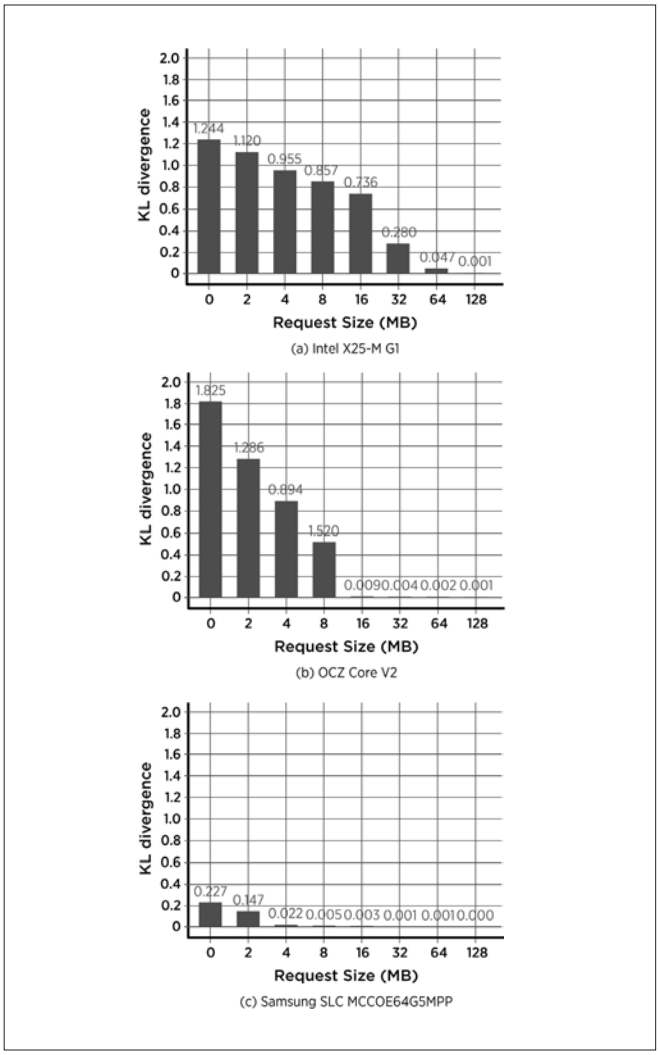


Figure 17. KL Divergence Values for Differenet Request Sizeds and SSDs. The optimal block size for INTEL SSD is 64MB, that for OCZ SSD is 16MB, and that for SAMSUNG SSD is 4MB.

5.2 System Design

In AHS systems, a segment is a basic unit for accessing videos. Accordingly, FlashStream organizes its storage structure at segment granularity. Figure 18 depicts the FlashStream architecture. The first cache layer is a RAM cache. The RAM buffer pool is an array

of memory blocks, and each block stores data for a video segment file. Because segments have different sizes, the size of each block in the RAM buffer pool is different.

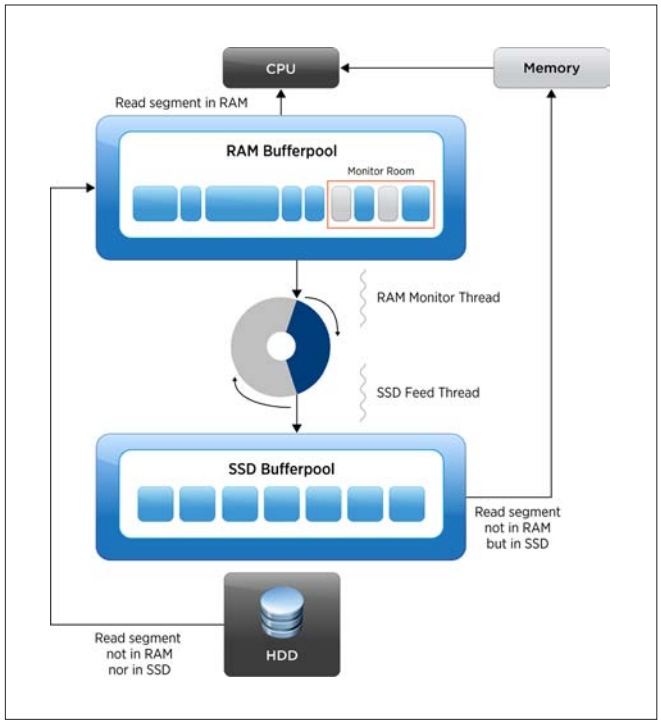


Figure 18. FlashStream Architecture. A finite-sized RAM buffer serves as the first-level cache of the server. SSD is the second-level cache. Upon a miss in the first-level cache, the data is served from the SSD (i.e., it is not copied into the first-level cache). A miss in both the caches results in reading the missing segment from the hard disk and into the first-level RAM buffer cache.

The second cache layer is a SSD cache. The SSD's address space is divided into equal-sized blocks (same as the optimal block size). All write operations to the SSD are requested with the optimal block size aligned on the block boundary. All segment requests that miss the two caches are served by disks. The segments read from the disks are pushed into the RAM buffer pool, and then they are returned to the CPU. The segments read from the SSD are not inserted into the first-level cache (RAM buffer pool) but are directly returned to the CPU (i.e., DMA'ed into its memory). In this way, we can maximally utilize the capacity of the first-level RAM cache and the second-level SSD cache because we avoid storing the same segment in both levels. One of our design goals is to minimize the I/O requests sent to the disks because the hard disk is the slowest element in the three-level hierarchy of FlashStream, and thus could hurt real-time streaming performance. By storing data either in the DRAM or the SSDs (but not both), FlashStream maximizes the cumulative cache hit ratio in the first two levels and minimizes the read requests sent to the disks.

5.2.1 RAM Buffer Pool

The RAM buffer pool is the first level in the three-level hierarchy of FlashStream. Segments cached in the buffer are ordered according to the replacement policy used. For example, when a LRU replacement policy is employed, the most recently used segment will be at the head of the buffer pool, and the least recently used segment will be at the tail. LRU is the default replacement policy for the RAM buffer pool because it is simple and works well in most cases.

However, it should be noted that other replacement policies (such as LRU-Min and Greedy-Dual-Size [GDS] [40]) can be easily used for the RAM buffer pool without affecting the design decisions in the other components of FlashStream. Whenever a new segment is inserted into the pool and the pool has insufficient space, segments at the end of the pool are evicted until the pool has space to host the new segment. A RAM monitor thread periodically monitors the tail part of the RAM buffer pool (Figure 18). This tail portion of the pool is called the monitor room. The size of the monitor room is a system-configuration parameter, and we use 10% of the total number of segments in the RAM buffer pool for the monitor-room size by default. When the thread finds segments in the monitor room that are not in the SSD cache, those segments are copied to a ring buffer. In Figure 18, such segments are shown as shaded squares in the monitor room. Segments that are in the ring buffer are candidates for being written into the second-level cache. The monitor thread simply throws away the segments in the monitor room to make room for new segments coming from the disk, even if they have not been copied into the ring buffer by the monitor thread. This is similar to the evict-ahead policy used in the ZFS file system. We assume that the video data is immutable and thus read-only (e.g., database of movies stored at Netflix). When a video object is updated (e.g., a new version of a movie is distributed to the CDN servers by the content provider), the cached copies of the segments pertaining to the old version of the video object will simply be removed from the memory hierarchy of FlashStream. This is an out-of-band administrative decision outside the normal operation of FlashStream and does not interfere with the evict-ahead policy for managing the buffer pool. The advantage of the evict-ahead policy is that it can avoid flash write operations from the critical path for handling cache miss (see Recommendation 2 in Section 4.3). The drawback of the evict-ahead policy is that there is no guarantee that the evicted segments will be written to the second-level cache (i.e., SSD). When a segment misses both the RAM and SSD caches, it is read from the disks and placed in the RAM buffer pool. A thread that is inserting a segment into the RAM buffer pool does not have to wait for a victim segment (in the monitor room) to be copied into the ring buffer (for writing eventually to the SSD). The hope is that the victim would have already been copied into the ring buffer by the monitor thread **ahead of time**.

5.2.2 SSD Buffer Pool

Segments that are to be written to the SSD are buffered in the ring buffer between the RAM buffer pool and the SSD buffer pool. While the RAM monitor thread fills up the ring buffer, a SSD feed thread consumes the ring buffer and writes data to the SSD in units of the optimal block size (Section 5.1). In addition, the SSD feed thread employs a simple admission mechanism. For a video streaming system, read operations have higher priority than write operations. Therefore, when the SSD is busy for serving read requests, the SSD feed thread should not make write requests to the SSD. Details of the admission mechanism are presented in Section 5.3.4. If the second-level cache (SSD) is full, the victim block chosen for eviction from the SSD is simply thrown away because the block is already present in the disk, and we are always dealing with read-only data in the storage server.

5.2.3 SSD Block-Replacement Policies

A LRU replacement policy and a Least Frequently Used (LFU) replacement policy are appropriate for evicting blocks from the SSD cache. However, there is an interesting design dilemma. Recall that our unit of management of the second-level SSD cache is the optimal block size of the SSD. Because segments can be of variable sizes, a single block in the SSD cache (of optimal block size) can hold multiple segment sizes. How do we update the LRU list of blocks when a segment that is a small part of a block is accessed? How do we maintain the LFU bookkeeping for multiple segments that might be in the same SSD block? We suggest three different replacement policies:

- Least Recently Used Block (LRU): In this policy, a segment access is considered the same as an access to the containing SSD block. Therefore, when any segment in a block is accessed, the block is moved to the front of an LRU list. When a block needs to be evicted, the last block in the list is chosen as a victim.
- Least Frequently Used Block on Average (LFU-Mean): This scheme keeps track of the access count for each segment. When a block needs to be evicted, the average access count of the segments in a block is calculated, and the block with the least average count is chosen as a victim.
- Least Frequently Used Block Based on Median (LFU-Median): Similar to the LFU-Mean policy, this scheme keeps track of the access count for each segment as well. Because the mean is not robust to outliers, this policy uses median of the access counts of the segments in a block.

The performance of these SSD block replacement policies is evaluated in Section 5.4.

5.2.4 Difference from ZFS

FlashStream employs an evict-ahead policy similar to the ZFS file system. However, there are fundamental differences between the two:

- ZFS uses the FIFO replacement policy for the SSD. The motivation is same as FlashStream; the FIFO replacement policy generates sequential writes to the SSD and avoids small random writes to the SSD. However, the FIFO replacement policy of ZFS shows a low hit ratio. In contrast, FlashStream employs different kinds of block replacement policies, and it shows a much higher hit ratio than ZFS. The results are presented in Section 5.4.2.
- ZFS does not differentiate the priority of writes and reads to/from the SSD because it is not designed for a video streaming system. Therefore, reads that would be hits in the SSD cache get queued up behind ongoing writes and miss the deadline that reads need to be serviced. In contrast, FlashStream uses an admission policy for writes to the SSD to give a higher priority for read requests.
- FlashStream decouples monitoring the RAM buffer and writing evicted segments in the ring buffer to the SSD. These two functions are intertwined in ZFS, and it could lead to lost opportunity. Recall that the evict-ahead policy used in ZFS will simply throw away pages from the RAM buffer even if they have not been written to the SSD. Thus, if the RAM insertion rate from the disk is faster than the rate at which evicted pages can be written to the SSD, a larger population of evicted pages will be thrown away. In contrast, in FlashStream, the monitor thread and the SSD feed thread cooperate through the ring buffer and thus there is an opportunity for more evicted segments to be cached in the SSD (despite the evict-ahead policy) even during periods when the RAM insertion rate is higher than the SSD write throughput. The amount of time taken to fill the SSD cache for both systems is presented in Section 5.4.2.

5.3 Implementation

In this section, we describe the details of FlashStream implementation. We built our FlashStream web server based on the pion-net [11] open source network library. FlashStream directly manipulates a flash memory SSD as a raw device. In addition, FlashStream bypasses the operating system's page cache and manages the RAM for caching segment files on its own. Each video segment is uniquely identified by a tuple: {video ID, segment number, bit rate} (see Table 2). The tuple represents the ID of a segment. Buffer Manager and SSD Manager are the two most import components, and their details are explained in the following subsections.

SEGMENT ID		
Video ID	Segment Number	Bit Rate

Table 2. Each video segment is uniquely identified by a tuple: {Video ID, Segment Number, Bit Rate}.

5.3.1 RAM Buffer Manager

A buffer manager manipulates the available RAM buffer for caching segments. Figure 19(a) shows data structures used by the buffer manager. It keeps a pool of cached segments in RAM. For fast lookup, a hash table (key-value store) is maintained in RAM. The key is a segment ID, and the value is a tuple: {memory address of the block in RAM, size of the block}. The segment list maintains a list of segment IDs for victim selection. The buffer manager uses a LRU replacement policy. A hit in the RAM results in the segment being moved to the head of the segment list. The last element of the segment list is a victim. When a new segment read from the disk is inserted to the RAM buffer pool, a new entry is inserted into the hash table, and the ID of the new segment is inserted at the head of the segment list. If needed, a victim is evicted from the buffer pool and the entry for the victim in the hash table is removed.

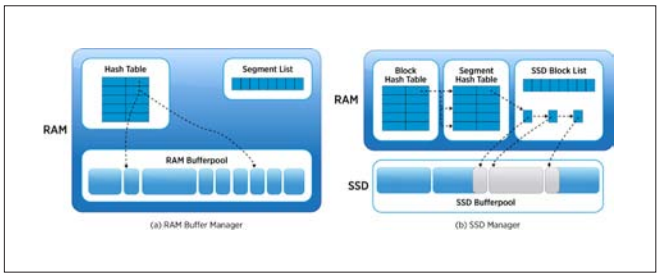


Figure 19. Data Structures Used by the RAM Buffer Manager and the SSD Manager. Replacement from the RAM buffer is in units of variable-sized video segments.

5.3.2 SSD Manager

A SSD manager manipulates the available SSD blocks for caching segments. Figure 19(b) shows the data structures used by the SSD manager. The SSD's address space is divided into equal-size blocks, and the block is the same as the optimal block size (see Section 5.1).

A block hash table (another key-value store) is maintained to look up the segments that are stored in a block. The key is a SSD block number, and the value is a list of segment IDs that are stored in the same block. A segment hash table is used to look up the logical block address of the SSD for a segment. When the segment size is larger than the size of a SSD block, the segment is stored using multiple blocks. In addition, a part of a segment can be written at the end of a block, and the rest can be written in the front of another block. We call such partial data of a segment a *chunk* of the segment. Therefore, the value field in the segment hash table is a list of tuples; each tuple contains {SSD block number, offset into the block, size of a segment chunk}. In Figure 19(b), the shaded blocks in the SSD buffer pool represent a single segment stored in multiple SSD blocks. A SSD block list maintains a list of block numbers for victim selection. The block hash table, the segment hash table, and the SSD block list data structures are stored in physical memory (i.e., RAM) for fast access.

5.3.3 SSD Block Data Layout

Flash memory SSDs provide capacity from hundreds of gigabytes to a few terabytes in a single drive these days. After a system crash or administrative downtime, warming up such a large-capacity SSD takes a long time. Because flash memory is nonvolatile, a warmed-up SSD can still serve as a second-level cache despite power failures as long as we have metadata for the segments stored in the SSD as well. To recover the stored data, FlashStream embeds metadata in the SSD blocks. Figure 20 shows the data layout in the SSD blocks. The first SSD block (block number 0) has a special record: the *super record*. The super record includes information about the block size and its checksum. Only the first SSD block has the super record; the other SSD blocks do not. Following the super record, a series of records are stored. Each record represents a chunk of a segment. The record consists of segment ID, number of chunks, chunk sequence number, chunk size, chunk data, and checksum. From the segment ID, the system can figure out the segment that the chunk belongs to. The number of chunks tells how many chunks the segment is divided into, and the chunk sequence number tells the order of the chunk in its segment. Checksum is used to verify the integrity of each record. After a system crash, by scanning all SSD blocks sequentially, FlashStream can reconstruct the in-memory data structures of the SSD manager (i.e., block hash table and segment hash table).

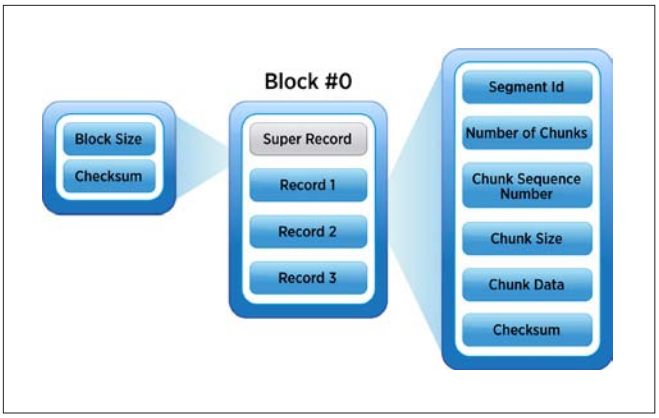


Figure 20. Data Structure of a SSD Block. A SSD block is composed of records, with each record containing the metadata as well as the data pertaining to the segment ID. Block #0 (the first block) of the SSD is special in that it contains a super record in addition to the data records.

5.3.4 Utilization-Aware SSD Admission

The SSD feed thread employs an admission mechanism to avoid writes to the SSD when the SSD is busy serving reads. To measure how busy the SSD device is, we use /proc/diskstats, wherein the Linux kernel reports information for each of the storage devices. /proc/diskstats provides the amount of time (in milliseconds) spent doing I/O for each storage device. We read the number at the beginning of a certain epoch and again at the end of the epoch. Because this is an accumulated number, the difference between the two readings gives the time spent in I/O for that device for the current epoch. The ratio of the time spent in I/O to the periodicity of the epoch serves as a measure of the storage-device utilization for that epoch. The SSD feed thread stops writing data to the SSD when the SSD utilization exceeds a threshold λ, and the default value for the threshold is 60%.

5.4 Evaluation

In this section, we evaluate the performance of the FlashStream system. We want to mimic the request pattern that is fielded and serviced by an HTTP server (i.e., a CDN server) that is serving video segments to distributed clients under the AHS mechanism. To this end, the workload offered to the FlashStream server is a large number of independent requests from clients. To measure the storage subsystem performance exactly and to avoid network subsystem effects, the workload-generator program and the FlashStream server run on the same machine, communicating via a loop-back interface. The machine has Xeon 2.26GHz Quad core processor with 8GB DRAM, and Linux kernel 2.6.32 is installed on it. We use a 7200RPM HDD, and the capacity of the HDD is 550GB. We evaluated five system configurations. In addition to our FlashStream system with three SSD cache replacement policies, we measured the performance of two state-of-the-art multitiered storage systems (that use flash memory caching): ZFS [12] and Flashcache [4].

CONFIGURATION	WEB SERVER	FILE SYSTEM
FlashStream(LRU)	FlashStream	EXT4
FlashStream (LFU-Mean)	FlashStream	EXT4
FlashStream (LFU-Median)	FlashStream	EXT4
ZFS	Apache	ZFS
Flashcache	Apache	EXT4

Table 3. Five System Configurations Used for Our Evaluation

Table 3 shows the web server and the file system that are used for each system configuration. The version of Apache HTTP Server is 2.2.14, and the version of ZFS is zfs-fuse 0.6.0-1. The flash memory SSDs we used for our experiments are listed in Table 4. Note that we intentionally use a very low-end SSD (SSD B) to show how well FlashStream works with low-end SSDs. In our evaluation, FlashStream and ZFS worked slightly better with SSD B, whereas Flashcache

worked better with SSD A. Therefore, we present only results of FlashStream and ZFS used with SSD B and results of Flashcache used with SSD A. We use only 75GB out of the total capacity for both SSDs for a fair comparison.

	SSD A	SSD B
Model	INTEL X25-M G1	OCZ Core V2
Capacity	80GB	120GB
4KB Random Read Throughput	15.5MB/s	14.8MB/s
4KB Random Write Throughput	3.25MB/s	0.02MB/s

Table 4. Flash Memory SSDs That Are Used for Our Experiments. SSD A shows significantly better random write performance than SSD B. Both SSDs have similar performance for small random reads. On the other hand, the different SSDs have significantly different small random write performance. We intentionally use a very low-end SSD (SSD B) to show how well FlashStream works with low-end SSDs.

5.4.1 Workload

Once again (as we did in Section 4.1.1) we choose the DASH dataset available in the public domain [31] for this performance study. As shown in Table 5, the dataset comprises six distinct video objects with five different bit rates for each video. However, for our experimentation we needed to create a larger dataset comprising 2,000 distinct videos. In other words, we needed to create a namespace of 2,000 distinct videos from the set of 6 videos. We generated the 2,000 videos by uniformly using the 6 videos and five bit rates. The total size of our dataset is 545GB. Figure 21 shows the zipf distribution of the 2,000 videos. Because we use five different bit rates, and videos are encoded with VBR, the segments of our generated dataset have very different sizes.

NAME	BIT RATES (KBPS)	LENGTH	GENRE
Big Buck Bunny	200, 400, 700, 1500, 2000	09:46	Animation
Elephants Dream	200, 400, 700, 1500, 2000	10:54	Animation
Red Bull Playstreets	200, 400, 700, 1500, 2000	97:28	Sport
The Swiss Account	200, 400, 700, 1500, 2000	57:34	Sport
Valkaama	200, 400, 700, 1400, 2000	93:05	Movie
Of Forest and Men	200, 400, 700, 1400, 2000	10:53	Movie

Table 5. Source DASH Dataset Used to Generate Our 2,000 Videos

In every t seconds (which is the inverse of the request rate), the workload generator selects a video object according to the zipf distribution. Next, it chooses a segment of the video object according to a uniform distribution; each segment of the video object has the same probability. The reason behind the uniform

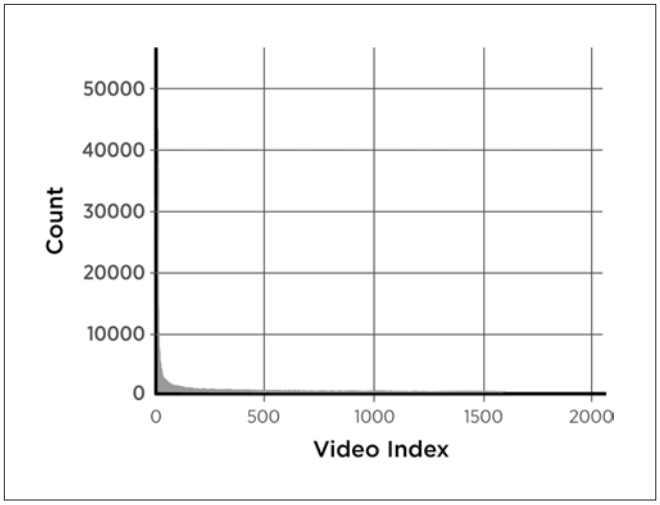


Figure 21. Distribution of Access Frequency of 2,000 Videos. The zipf parameter is 0.2. Top 400 (20%) popular videos account for 65.9% of total requests.

distribution is explained in Section 4.1.1. We use 10-second-long segments and measure the segment miss ratio against different request rates, similar to Section 4.1.1.

5.4.2 Performance Comparison

In this section, we measure the maximum request rate that the five systems can support when the cache is warmed up. This is the best performance each of the systems can provide. We have fully filled up the SSD cache by running the workload generator for a sufficient amount of time before each measurement. FlashStream (regardless of replacement policy) takes 94 minutes to fill up the 75GB SSD cache. ZFS takes 1,200 minutes (20 hours) and Flashcache takes 1,320 minutes (22 hours) for the same amount of SSD cache. We assume the QoS requirement for video streaming is 0% segment miss ratio. By increasing the request rate gradually, we measure the segment miss ratio for each request rate until the segment miss ratio is higher than 0%. In this way, we can get the maximum request rate for 0% segment miss ratio. We run each measurement with a different request rate for an hour.

Figure 22(a) shows that FlashStream performs two times better than ZFS. Different SSD cache-replacement policies of FlashStream perform similarly, with LRU slightly better than the other two. ZFS is able to support a maximum of 26 requests per second, and Flashcache cannot support even 1 request per second.

Next, we measure the SSD cache hit ratio of the five systems with a warmed-up cache when the request rate is the maximum request rate that the systems can support. Each measurement is run for an hour. Figure 22(b) shows that FlashStream with a LRU SSD cache-replacement policy has the best hit ratio. The other two policies (i.e., LFU-Mean and LFU-Median) show a slightly lower hit ratio than LRU.

ZFS's lower maximum request rate compared to FlashStream correlates directly with ZFS's lower SSD cache hit ratio. Page reads that miss the SSD cache of ZFS go to disks and make the disks overloaded, and the reads from the disks miss their deadline. This result tells us that, for the SSD cache, block-level replacement policies (which FlashStream uses) are able to achieve a higher

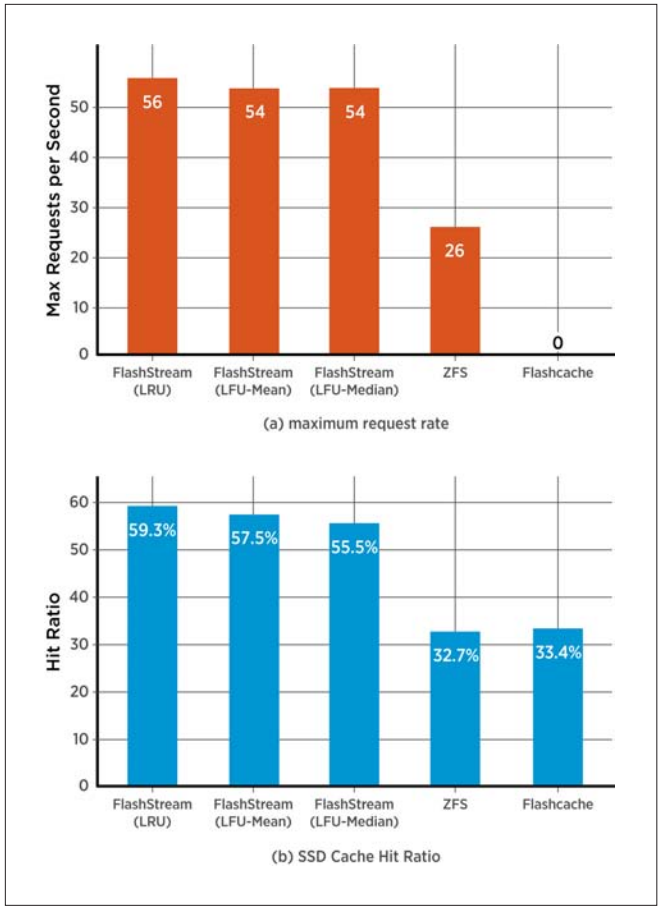


Figure 22. Maximum Request Rate and SSD Cache Hit Ratio of the Five Different System Configurations with a Warmed-Up Cache for 0% Segment Miss Ratio. FlashStream performs two times better than ZFS.

hit ratio than FIFO replacement policy (which ZFS uses), and results in a higher maximum request rate. The reason for the poor performance of Flashcache is that SSD write operations are in the critical path of serving read operations that miss the SSD cache. Thus, though the observed hit ratio for Flashcache is 33.4%, it does not translate to performance (as measured by the maximum request rate) due to the fact that SSD cache reads could be backed up behind long-latency SSD cache write operations. FlashStream and ZFS systems address this problem by the evict-ahead mechanism (see Section 5.2.1). For a more detailed analysis of the poor performance of ZFS and Flashcache, refer to Section 4.2.

5.4.3 Effect of Block Size

In this section, we measure how the choice of the SSD block size affects FlashStream performance. We run FlashStream with a cold cache for an hour using SSD B. Even though the SSD cache is empty, for this experiment, we intentionally make FlashStream return random free blocks (not in a sequential order) when the SSD feed thread needs to write a block of data to the SSD. We do this because we want to see quickly the effect of the SSD writes with random offsets without filling up the SSD cache. This is for a quick measurement and it does not harm the correctness of the results. As shown in Figure 17, the optimal block size of SSD B is 16MB. Figure 23(a) shows read throughput, write throughput, and utilization (top to bottom) of the SSD for an hour. From the write-throughput graph (middle), we notice that the system gradually fills segments into the SSD cache. The read throughput graph (top) shows that the read throughput increases as segment requests hit the SSD cache. The utilization graph (bottom) tells us that the SSD device is not overloaded during this period.

Figure 23(b) presents the same SSD's read throughput, write throughput, and utilization when the block size is 4KB. SSD B is a low-end SSD that shows very low performance with small random writes (see Table 4). Due to the write amplification triggered by the small block writes with random offsets, the SSD is overloaded during the whole period (100% utilization), and hence provides very low read and write throughput.

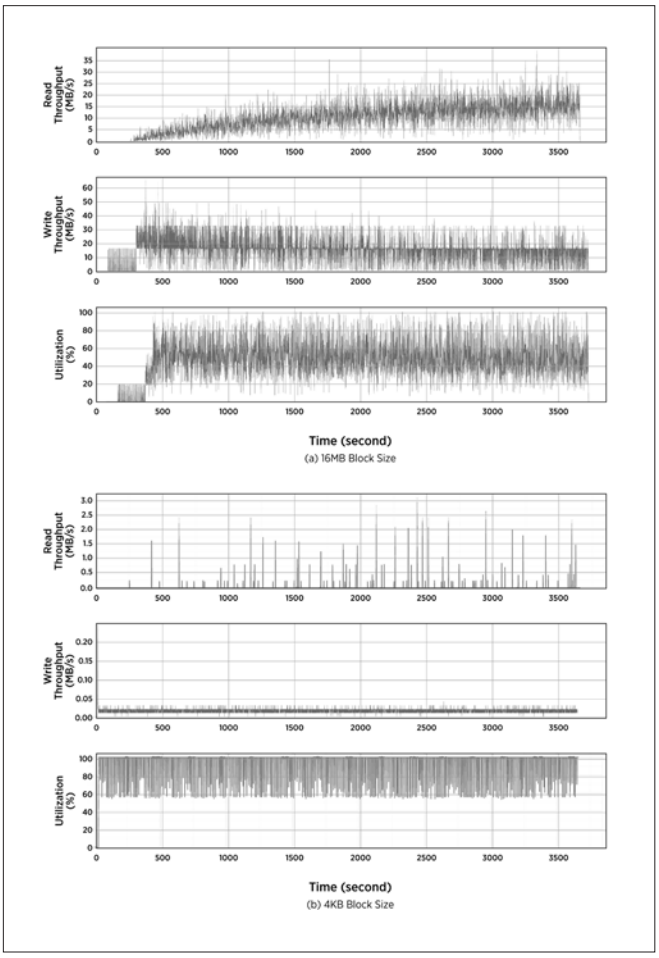


Figure 23. Effect of Different Block Sizes on Read Throughput, Write Throughput, and Utilization of the SSD (Second-Level Cache of Flashstream). When the block size matches the optimal block size (Figure 23 (a)), the SSD is not overloaded (i.e., there is no write amplification), and the system fills segments gradually and serves the segments that hit the SSD. When the block size is 4KB, (Figure 23(b)), the SSD is overloaded (due to write amplification) and shows very low read and write throughput.

When the block size is 16MB, during the one-hour period, 70.4% of the total capacity of the SSD is filled, and the SSD hit ratio is 34.4%. In contrast, when the block size is 4KB, only 0.08% of the total SSD capacity is written, and the SSD hit ratio is 0.65%. To make matters worse, 3% of the read requests that hit the SSD miss their deadline due to the poor read throughput. This experiment underscores the importance of using the optimal block size for the SSD to achieve good performance.

5.4.4 Read Throughput Consistency

The important attribute of FlashStream is consistent read throughput from the storage system. Figure 24 shows the variance of storage read throughput for FlashStream and Flashcache. The cache is warmed up before the measurement. We run the workload generator for an hour with 50 requests per second for both systems and measure total read throughput of all storage devices (i.e., HDDs and SSDs). A throughput sample is measured with a 1-second window. For FlashStream, the median read throughput is 50.09MB/s, the standard deviation is 13.68MB/s, and 50% of throughput samples are between 40MB/s and 60MB/s. For Flashcache, the median read throughput is 0.62MB/s and the standard deviation is 21.64MB/s. This result shows the superiority of FlashStream for providing consistent storage throughput for read requests.

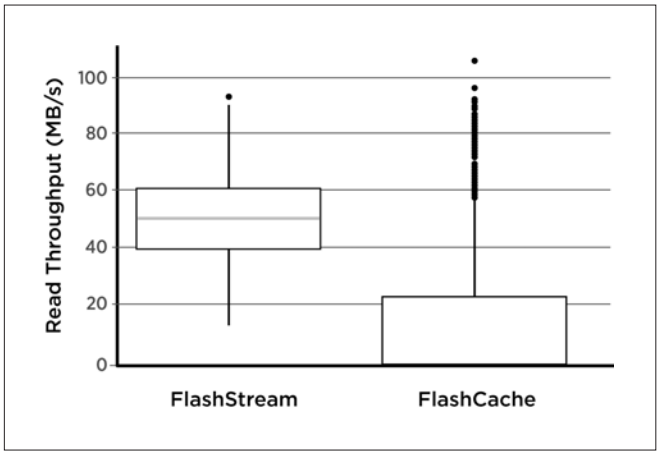


Figure 24. Read Throughput Variance. FlashStream has a median of 50.09MB/s and a standard deviation of 13.68MB/s. Flashcache has a median of 0.62MB/s and a standard deviation of 21.64MB/s.

5.4.5 Energy Efficiency

In this section, we compare the energy efficiency for three different systems: FlashStream and two "traditional" systems (one for more DRAM and another for more disks). Table 6 shows the hardware configuration of the three systems. FlashStream has 2GB DRAM, 100GB SSD, and a single 7200RPM 1 TB HDD. Traditional A has 12GB DRAM and a single 7200RPM 1TB HDD. Traditional B has 2GB DRAM and two 7200RPM 1TB HDDs striped per Linux's software RAID-0 configuration. Compared to FlashStream, Traditional A uses more DRAM for caching instead of SSDs, whereas Traditional B uses more HDDs for more disk throughput via parallel access. All three systems have the similar total capital cost according to the capital cost of devices in Table 7 (Traditional B is $10 more expensive than the other two). Apache HTTP Server is used for traditional systems. Table 7 shows the energy cost of each device in the active state. Each DRAM module in our test machine has 2GB; 12GB DRAM corresponds to 6 DIMMs. For FlashStream, we measure the maximum request rate after the SSD is filled up. For traditional systems, we warm up for an hour, which is sufficiently long to fill up the given DRAM. We use the same workload as in Section 5.4.1.

Table 6 shows that FlashStream achieves the best energy efficiency. FlashStream serves 64 requests per second with 0% segment miss ratio, and its energy efficiency is 2.91 requests per joule. For traditional configurations, using additional HDD (i.e., Traditional B) is better than using more DRAM (i.e., Traditional A). FlashStream is 94% more energy-efficient than Traditional B. Moreover, Traditional B wastes disk capacity because it uses more HDDs only for more throughput; the dataset size is 545GB whereas Traditional B has 2 TB HDDs in total. In addition, we ignore the cooling cost and the cost associated with rack space in a data center. FlashStream that uses less DRAM and HDDs, but more flash-based SSDs would generate less heat and occupy less volume in a rack. We do not consider these benefits in this analysis.

CONFIGURATION	DRAM	SSD	HDD	RPS	REQUESTS/ JOULE
FlashStream	2GB	100GB	1	64	2.91
Traditional A	12GB	-	1	34	0.57
Traditional B	2GB	-	2	48	1.50

Table 6. Three System Configurations for Energy Efficiency Comparison. SSD is a low-end SSD, and HDD is 7200RPM. RPS represents the maximum request rate.

	CAPITAL COST	ENERGY COST
DRAM	$8/GB	8W/DIMM
Low-end SSD	$0.8/GB	2W/drive
7200RPM 1TB HDD	$90/drive	12W/drive

Table 7. Capital Cost and Energy Cost of Devices. Data comes from the specification of commodity DRAM, SSDs, and HDDs that are commercially available [10].

5.5 Summary

In this section, we proposed a set of guidelines for effectively incorporating flash memory SSDs for building a high-throughput and power-efficient multitiered storage system for AHS. We implemented FlashStream based on these guidelines and evaluated its performance using realistic workloads. We compared the performance of FlashStream to two other system configurations: Oracle's ZFS, and Facebook's Flashcache. Similar to FlashStream, the two configurations also incorporate flash memory SSDs in their respective storage hierarchies. FlashStream performs twice as well as its closest competitor, the ZFS configuration, using segment miss ratio as the figure of merit for comparison. In addition, we compared FlashStream with a traditional two-level storage architecture (DRAM + HDDs), and showed that, for the same investment cost, FlashStream provides 33% better performance and 94% better energy efficiency.

6. Related Work

In recent years, there have been many different proposals to incorporate flash memory in computer systems. In this section, we summarize prior system studies related to flash memory integration.

6.1 Multitiered Storage Systems

Flash memory provides faster read latency and consumes less energy than magnetic disks. However, its random writes can be slower than the disks, and it is more expensive than the disks. Enterprise storage systems simultaneously require high performance, low energy consumption, and large capacity. Therefore, tiering flash

memory between DRAM and the disks, and caching hot data in flash memory, are promising ideas to achieve these requirements. Multitiered storage systems have been designed in different ways depending on the application workload and system requirements. FlashTier [45] and Flashcache [4] are block device systems; applications and file systems that sit above the block device layer thus automatically reap the benefits of the flash without any modification. Solaris ZFS [12] suggests a multitiered storage system in the form of a file system, and applications accessing files on ZFS can get the benefit of flash memory without modification. In addition, there have been applications that directly handle DRAM, flash memory, and disks to achieve application-specific requirements. FlashStore [23] is a high-throughput persistent key-value storage using flash memory as a cache between the RAM and the hard disk. FlashStore is designed for applications that need a persistent object store, and it shows better performance and energy efficiency than BerkeleyDB. Lee et al. [34] [33] have researched ways to improve database performance with flash memory SSDs. The authors claim that a single enterprise-class SSD can be on par with or far better than a dozen hard drives with respect to transaction throughput, cost-effectiveness, and energy consumption. Do et al. [24] have proposed three different design alternatives that use a SSD to improve the performance of a database server's buffer manager. They empirically have shown that they could speed up three to nine times more than the default HDD configuration depending on system parameters. Kgil et al. [29] have studied energy-efficient web servers using flash memory as an extended system memory. Their work is based on simulations with text and image workloads. Singleton et al. [47] have shown how much power can be saved when flash memory is used as a write buffer along with hard disks for mobile multimedia systems.

6.2 Flash-Only Key-Value Storage Systems

FAWN [16] is a distributed key-value storage system for low-power and data-intensive computing. FAWN couples low-power embedded CPUs to small amounts of local flash storage, and balances computation and I/O capabilities to enable efficient, massively parallel access to data. The authors of FAWN have shown that FAWN can handle roughly 350 key-value queries per joule, which is two orders of magnitude more than a disk-based system.

6.3 Extended Memory

A flash SSD is not used only as fast storage. It can be exploited as slow (but large-capacity) memory-extending DRAM. FlashVM [44] is a system that proposes using a flash SSD as a dedicated swap device that provides hints to the SSD for better garbage collection by batching writes, erases, and discards. SSDAlloc [17] is a user-level runtime library that allows developers to treat flash SSDs as an extension of the DRAM in a system. SSDAlloc moves the SSD upward in the memory hierarchy, making it usable as a larger, slower form of memory instead of a storage alternative for the hard disk. Using these techniques, applications can transparently extend their memory footprint to a few terabytes without any restructuring, far exceeding the DRAM capacities of most servers used in data centers.

The focus of this paper is different from these other research endeavors. FlashStream shows how to construct a high-performance and energy-efficient multitiered storage system for AHS through the judicious use of flash memory SSDs as an intermediate level in the storage hierarchy. The most important requirement for video streaming is a fast and consistent read latency when accessing video data, due to the real-time nature of video streaming. FlashStream is optimized to satisfy the requirement even when used with low-end flash SSDs, and hence it could achieve both high performance and low energy consumption.

7. Conclusions

The need for high–bit rate on-demand video content on the Internet is exploding. To satisfy reliable on-demand video streaming, a video service provider should provide a significant amount of storage bandwidth. Support for such a huge amount of storage bandwidth imposes a significant cost burden on the service provider. Such a cost burden is aggravated in the AHS paradigm because AHS systems rely on overprovisioned CDN resources for reliable video delivery over the Internet.

In this paper, we first show that HDDs are not ideal for serving video content in AHS systems: The video segment size can be very small and diverse in AHS systems, and HDDs are very slow for reading such small files. On the other hand, flash memory SSDs are very fast for reading small files and consume a lot less energy than HDDs. However, flash-based SSDs have smaller capacity than HDDs for a given cost. Therefore, multitiered storage in which flash-based SSDs sit in the middle between DRAM and HDDs has a great potential for AHS that needs large capacity and large bandwidth at the same time. Next, we evaluate our hypothesis using state-of-the-art multitiered storage systems with AHS workloads. Contrary to our expectations, such systems show less or no performance advantage for AHS compared to traditional two-tier storage systems. We analyze the reasons for the counterintuitive results and suggest design guidelines for constructing a multitiered storage system for AHS. Finally, we propose our FlashStream system, which is designed and implemented based on the guidelines. For the same storage cost, FlashStream provides 33% better performance and 94% better energy efficiency compared to the two-tier storage architecture.

There are several avenues for future research building on our study of a multitiered storage system. NAND flash memory is the first nonvolatile memory technology that is currently widely used in computer systems. In recent years, there have been many different proposals for ways to incorporate flash memory in computer systems. Flash memory has impacted greatly both embedded systems and high-performance systems. Our own studies and related work help surface research issues that new nonvolatile memory will bring to the forefront from the point of view of both the computer architecture and system software structure. With the advent of *storage class memory* (SCM) the distinction between primary (physical) memory and storage is getting blurred. NAND flash memory has been around for a while and has been used inside SSDs quite successfully as a viable alternative to HDDs.

However, NAND flash has serious performance and reliability problems (e.g., poor random write, wear-out issue). Although NAND flash is also a type of SCM, the exciting change in recent times is the advent of newer technologies such as *phase change memory* (PCM) [46] and *spin torque transfer* (STT) magnetic RAM [21]. With improvements in process technology for making PCMs, it is expected to scale much better than NAND flash, well into the single-digit nanometer feature sizes. PCM has read and write latencies that are about two orders of magnitude lower than NAND flash (i.e., closer to DRAM latencies). PCM also has better durability than NAND flash: a few tens of millions writes per cell, as opposed to only a few thousand writes for NAND flash. Perhaps what makes PCM so attractive is the fact that it can be used to design byte-addressable persistent memory, making it a viable competitor to DRAM. At this point in time, PCM is the most mature of the newer SCM technologies. However, STT-RAM has low access latencies (< 20ns) and is even more promising as a future competitor to DRAM. Perhaps what is more exciting is the fact that computer architects have been eyeing these technologies from the point of view of replacing DRAM with such nonvolatile counterparts [41] [32] [20]. Until recently, for over 50 years, system research has assumed two-level memory and storage: fast volatile memory and slow nonvolatile storage devices. The performance of storage increased, but the gap between memory and storage seldom diminished before advent of NAND flash–based SSDs. But, what if those assumptions are no longer valid, so that it is possible to reveal fast nonvolatile memory instead of two separate memory-hierarchy structures? Instead of using nonvolatile memory as a fast storage device that replaces mechanical HDD drives, how can we use new persistent memory in more revolutionary ways? This change, more than any other aspect of computer organization, has the potential of requiring a revolutionary approach to the way system software is built—in particular, the operating system itself. The entire system software stack should be redesigned including virtual memory, file systems, process schedulers, and even protection mechanisms.

References

1. Apache HTTP Server. http://httpd.apache.org.
2. blktrace. http://linux.die.net/man/8/blktrace.
3. Ext4 file system. https://ext4.wiki.kernel.org.
4. Flashcache. http://www.facebook.com/note.php?note_id=388112370932.
5. HDD Technology Trends. http://www.storagenewsletter.com/news/disk/hdd-technology-trends-ibm.
6. HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt.
7. Hulu viewers. http://www.comscore.com/Press_Events/Press_Releases/2011/12/comScore_Releases_November_2011_U.S._Online_Video_Rankings.
8. ISO/IEC DIS 23009-1.2. Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats.
9. Netflix traffic. http://www.techspot.com/news/46048-netflix-represents-327-of-north-americas-peak-web-traffic.html.
10. Newegg. http://www.newegg.com.
11. pion-net. http://pion.sourceforge.net/.
12. ZFS. https://docs.oracle.com/cd/E19253-01/819-5461/zfsover-2/.
13. ZFS L2ARC. https://blogs.oracle.com/brendan/entry/test.
14. Adhikari, V. K., Guo, Y., Hao, F., Varvello, M., Hilt, V., Steiner, M., and Zhang, Z.-L. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In INFOCOM (2012), IEEE, pp. 1620–1628.
15. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. Design tradeoffs for SSD performance. In ATC'08: USENIX 2008 Annual Technical Conference (Berkeley, CA, USA, 2008), USENIX Association, pp. 57–70.
16. Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. FAWN: a fast array of wimpy nodes. In Proceedings of the 22nd ACM Symposium on Operating Systems Principles (Big Sky, MT, USA, October 2009), ACM.
17. Badam, A., and Pai, V. S. SSDalloc: hybrid SSD/RAM memory management made easy. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (March 2011), NSDI'11.
18. Barbir, A., Cain, B., Nair, R., and Spatscheck, O. Known content network (CN) request-routing mechanisms. RFC 3568, http://tools.ietf.org/html/rfc3568.
19. Begen, A. C., Akgul, T., and Baugher, M. Watching video over the web: Part1: Streaming protocols. IEEE Internet Computing 15, 2 (2011), 54–63.
20. Caulfield, A. M., Coburn, J., Mollov, T., De, A., Akel, A., He, J., Jagatheesan, A., Gupta, R. K., Snavely, A., and Swanson, S. Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10) (2010), IEEE Computer Society, Washington, DC, USA, pp. 1–11.
21. Chung, S., Rho, K.-M., Kim, S.-D., Suh, H.-J., Kim, D.-J., Kim, H.-J., Lee, S.-H., Park, J.-H., Hwang, H.-M., Hwang, S.-M., Lee, J.-Y., An, Y.-B., Yi, J.-U., Seo, Y.-H., Jung, D.-H., Lee, M.-S., Cho, S.-H., Kim, J.-N., Park, G.-J., Jin, G., Driskill-Smith, A., Nikitin, V., Ong, A., Tang, X., Kim, Y., Rho, J.-S., Park, S.-K., Chung, S.-W., Jeong, J.-G., and Hong, S.-J. Fully integrated 54nm STT-RAM with the smallest bit cell dimension for high density memory application. In Electron Devices Meeting (IEDM), 2010 IEEE International (December 2010), pp. 12.7.1 –12.7.4.

22. Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2013–2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

23. Debnath, B., Sengupta, S., and Li, J. Flashstore: High throughput persistent key-value store. In Proceedings of the 36th International Conference on Very Large Data Bases (Singapore, September 2010).

24. Do, J., Zhang, D., Patel, J. M., DeWitt, D. J., Naughton, J. F., and Halverson, A. Turbocharging DBMS buffer pool using SSDs. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (2011), SIGMOD '11, pp. 1113–1124.

25. Gray, J., and Fitzgerald, B. Flash disk opportunity for server-applications. http://www.research.microsoft.com/~gray (January 2007).

26. Hwang, C.-G. Nanotechnology enables a new memory growth model. In Proceedings of the IEEE 91(11) (November 2003), pp. 1765–1771.

27. Intel Corporation. Understanding the Flash Translation Layer (FTL) Specification. White Paper, http://www.embeddedfreebsd.org/Documents/Intel-FTL.pdf, 1998.

28. Kawaguchi, A., Nishioka, S., and Motoda, H. A flash-memory based file system. In USENIX Winter (1995), pp. 155–164.

29. Kgil, T., and Mudge, T. Flashcache: a NAND flash memory file cache for low power web servers. In Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (2006), CASES '06, pp. 103–112.

30. Kullback, S., and Leibler, R. A. On information and sufficiency. Annals of Mathematical Statistics 22, 1 (March 1951), 79–86.

31. Lederer, S., Müller, C., and Timmerer, C. Dynamic adaptive streaming over HTTP dataset. In Proceedings of the Third Annual ACM Conference on Multimedia Systems (Chapel Hill, North Carolina, USA, February 2012), MMSys '12, pp. 89–94.

32. Lee, B. C., Ipek, E., Mutlu, O., and Burger, D. Architecting phase change memory as a scalable DRAM alternative. SIGARCH Comput. Archit. News 37, 3 (June 2009), 2–13.

33. Lee, S.-W., Moon, B., and Park, C. Advances in flash memory ssd technology for enterprise database applications. In Proceedings of the ACM SIGMOD (June 2009), pp. 863–870.

34. Lee, S.-W., Moon, B., Park, C., Kim, J.-M., and Kim, S.-W. A case for flash memory SSD in enterprise database applications. In Proceedings of the ACM SIGMOD (June 2008), pp. 1075–1086.

35. Leventhal, A. Flash storage memory. Communications of the ACM 51, 7 (July 2008), 47–51.

36. Nair, T. R. G., and Jayarekha, P. A rank based replacement policy for multimedia server cache using zipf-like law. Journal of Computing 2, 3 (2010), 14–22.

37. Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., and Rowstron, A. Migrating server storage to SSDs: Analysis of tradeoffs. In Proceedings of the ACM EuroSys (Nuremberg, Germany, April 2009).

38. Park, C., Cheon, W., Kang, J., Roh, K., Cho, W., and Kim, J.-S. A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications. Trans. on Embedded Computing Sys. 7, 4 (2008), 1–23.

39. Partridge, C., Mendez, T., and Milliken, W. Host anycasting services. RFC 1546, http://tools.ietf.org/html/rfc1546.

40. Podlipnig, S., and Böszörmenyi, L. A survey of web cache replacement strategies. ACM Comput. Surv. 35, 4 (December 2003), 374–398.

41. Qureshi, M., et al. Scalable high performance main memory system using phase-change memory technology. In ISCA-36 (2009).

42. Ryu, M., Kim, H., and Ramachandran, U. Why are state-of-the-art flash-based multi-tiered storage systems performing poorly for HTTP video streaming? In Proceedings of the 22nd SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (Toronto, Ontario, Canada, June 2012), NOSSDAV '12.

43. Ryu, M., and Ramachandran, U. FlashStream: A multi-tiered storage architecture for adaptive HTTP streaming. In Proceedings of the 21st ACM International Conference on Multimedia (Barcelona, Catalunya, Spain, October 2013).

44. Saxena, M., and Swift, M. M. FlashVM: virtual memory management on flash. In Proceedings of the 2010 USENIX Conference (June 2010), USENIX ATC'10.

45. Saxena, M., Swift, M. M., and Zhang, Y. FlashTier: a lightweight, consistent and durable storage cache. In Proceedings of the 7th ACM European Conference on Computer Systems (April 2012), EuroSys '12, pp. 267–280.

46. Servalli, G. A 45nm generation phase change memory technology. In Electron Devices Meeting (IEDM), 2009 IEEE International (December 2009), pp. 1 –4.

47. Singleton, L., Nathuji, R., and Schwan, K. Flash on disk for low-power multimedia computing. In Proceedings of the ACM Multimedia Computing and Networking Conference (January 2007).

48. Stockhammer, T. Dynamic adaptive streaming over HTTP: standards and design principles. In Proceedings of the Second Annual ACM conference on Multimedia Systems (2011), MMSys '11, pp.

Reducing Cache-Associated Context-Switch Performance Penalty Using Elastic Time Slicing

Nagakishore Jammula
Georgia Institute of Technology
nagakishore@gatech.edu

Moinuddin Qureshi
Georgia Institute of Technology
moin@ece.gatech.edu

Ada Gavrilovska
Georgia Institute of Technology
ada@cc.gatech.edu

Jongman Kim
Georgia Institute of Technology
jkim@ece.gatech.edu

Abstract

Virtualization enables a platform to have an increased number of logical processors by multiplexing the underlying resources across different virtual machines. The hardware resources are time-shared not only among different virtual machines (VMs), but also among different workloads of the same VM. An important source of performance degradation in such a scenario are the cache-warmup penalties that a workload experiences when it's scheduled, because the working set belonging to the workload gets displaced by other concurrently running workloads. We show that a VM that time-switches among four workloads can cause some of the workloads a slowdown of as much as 54%. However, such performance degradation depends on the workload behavior, with some workloads experiencing negligible degradation and some severe degradation.

We propose *Elastic Time Slicing* (ETS) to reduce the context-switch overhead for the most-affected workloads. We demonstrate that by taking the workload-specific context-switch overhead into consideration, the CPU scheduler can make better decisions to minimize the context-switch penalty for the most-affected workloads, thereby resulting in substantial performance improvements. ETS enhances performance without compromising on response time, thereby achieving dual benefits. To facilitate ETS, we develop a low-overhead hardware-based mechanism that dynamically estimates the sensitivity of a given workload to context switching. We evaluate the accuracy of the mechanism under various cache-management policies and show that it is very reliable. Context-switch–related warmup penalties increase as optimizations are applied to address traditional cache misses. For the first time, we assess the impact of advanced replacement policies and establish that it is significant.

1. Introduction

Virtualization enables sharing of hardware resources by multiple guest operating system (OS) instances. The resources are shared not only among different VMs, but also among different workloads of the same VM. To facilitate high utilization through consolidation, the system must support a large number of workloads. Some systems adopt coarse-grained division at the level of single cores, and others employ fine-grained division through time-sharing a core among workloads [1]. The latter phenomenon is referred to as *multitasked virtualization*. Factors such as cost, security, and system-management convenience lead to more workloads per system. The transition from dedicated workstations to virtualized desktop infrastructure environments is another trend in this direction.

In a virtualized environment with multiple workloads per VM, the time slice allocated to a VM is split equally among the constituent workloads [1]. As a result, each workload obtains a share of the time slice allotted to the VM, which is inversely proportional to the number of workloads. Such an aggressively multitasked environment serves as the basis for our work. Multitasked virtualization affects performance in two ways: (1) direct overhead incurred to switch among the workloads and (2) indirect overhead incurred due to the displacement of the system state. The second factor contributes significantly to the performance degradation and can be further viewed as composed of multiple components: lost register, translation look-aside buffer, branch predictor, and cache states. Among these components, the major overhead is due to the displaced state in the last-level cache (LLC) [1] and is the focus of this work. We designate the additional cache misses suffered due to a context-switch (CS) event as CS *misses*. The performance penalty associated with CS misses is severe in the case of multitasked virtualization, due to an additional degree of multitasking above and beyond the OS-level multitasking.

Modern computer systems feature large-LLC and long-latency main memory. When run on such systems, memory-intensive tasks cache a large volume of data in the LLC. We use the terms *workload*, *task*, and *application* as synonyms. After running a task of interest for the duration of its time-slice value, when the CPU scheduler context-switches to a different task or a set of different tasks, the cache lines belonging to the former are replaced by those of the latter. Depending on the memory-access behavior of the intervening tasks, when the task of interest gets a schedule on the processor again, it is likely to encounter a partially or completely cold cache. Depending on the memory-reuse behavior of the task of interest, its performance could be affected across the spectrum ranging from no or slight degradation to significant degradation. Some tasks experience only slight degradation because sometimes caches hold data irrelevant to future accesses [2].

We illustrate the variation in CS penalty across applications by using an example. Figure 1 shows the impact of CS events on the performance of two different applications. On a CS event, the cache warmup penalty is minimal for application (a) and significant

for application (b). Whereas (a) is not sensitive to CS events, (b) is highly sensitive. Even though the complete cache state is lost in case of both applications (a) and (b) on a CS event, (a) suffers only minor performance degradation because its data reuse is low. (b) suffers significant performance degradation because its data reuse is high. In the following section, we use actual data to show that different tasks suffer from CS misses differently. For a task that suffers from CS misses significantly, a small time-slice value causes the task to experience CS events and CS misses more times than a large time-slice value. This phenomenon translates to an increase in the execution time of the task and a corresponding increase in the energy consumed across the entire system. The problem can be addressed by allocating fewer but longer time slices to the most-affected tasks (as illustrated in Figure 1(c)).

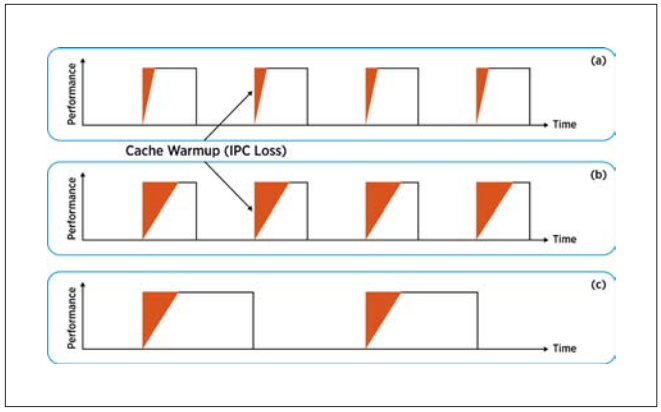


Figure 1. (a) When CS penalties are small, using short time slices does not cause any noticeable overhead. (b) For some workloads, short time slices can cause significant slowdown. (c) Only for such workloads is using longer and infrequent time slices desirable.

In this paper, we propose ETS to reduce the CS miss penalty. Whereas a uniform time-slicing (UTS) CPU-scheduling algorithm allocates time slices of equal duration to all tasks irrespective of their specific CS miss behavior, an ETS CPU-scheduling algorithm analyzes the CS miss behavior of the tasks and allocates fewer but longer time slices to those tasks that suffer significant performance degradation due to CS events. Performance penalty due to CS events can be naïvely addressed by allocating 10ms time slices to all tasks. 10ms is the default time-slice value allocated by the Linux OS. However, this solution suffers from high latency or response time between consecutive schedules, as depicted in Figure 2. In contrast, a UTS algorithm with 2.5ms time slices achieves low latency between consecutive schedules, but it suffers from low performance. 2.5ms is obtained by dividing 10ms equally among four tasks of a VM. Our ETS algorithm combines the best of both worlds and offers high performance (within 4% of UTS-10) as well as low latency (similar to UTS-2.5).

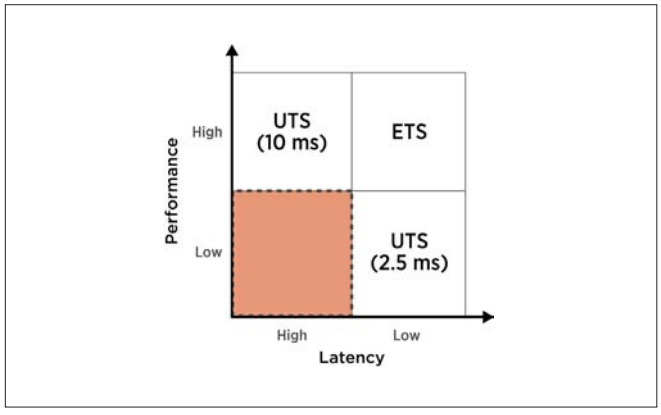


Figure 2. ETS provides the performance benefits of UTS with 10ms time slices as well as the latency benefits of UTS with 2.5ms time slices.

Enabling ETS requires dynamically estimating the extent to which a task suffers from CS misses. We develop a low-cost hardware-based Monte Carlo mechanism to estimate the cost of a CS event in terms of the number of CS misses suffered. The CS cost estimator works reliably under various cache-management policies because it is based on sampling of actual CS miss information. It facilitates incorporating the information about CS miss behavior into the design of a CPU-scheduling algorithm and exploiting the potential of such an enhanced CPU scheduler.

Most solutions that attempt to improve the cache hit rate by addressing the traditional cache misses (such as those due to capacity, conflict, coherence, and replacement) accentuate the problem of CS misses. These include: increasing the capacity of cache, employing compression in cache, prefetching lines into cache, improving the replacement algorithm, and so on. The number of CS misses tends to increase with increase in cache capacity (section 6.2) and improvement in replacement algorithm (section 6.1), thus worsening the problem. This paper shows that as systems optimize cache organization, addressing the problem associated with CS misses becomes more important, and a scheme like ETS becomes even more relevant.

2. Motivation

The locality properties of applications vary, and hence losing the cache state due to context switch can lead to variation in performance degradation for different applications. To demonstrate this, we conducted an experiment by reducing the allocated time-slice value. Figure 3 shows the variation in slowdown (measured in terms of cycles per instruction [CPI]) for SPEC CPU2006 benchmarks as the assigned time-slice value is reduced from 10ms. We flush the caches after each time slice to emulate a CS event. The rationale behind flushing the caches on a CS event will be described shortly. The parameters of the simulation infrastructure used to generate the results provided in Figure 3, and the basis for the choice of the parameters, are provided in section 4. Here, we capture the important aspects in order to enable comprehension of Figure 3. The results correspond to a LLC capacity of 2MB. We consider a processor running at a frequency of 4GHz. On such a processor, 10 million elapsed cycles correspond to an execution time of 2.5ms. The Y-axis represents the CPI corresponding to the execution of 500 million instructions. The labels 2.5ms and 5ms correspond to the cases when the VM comprises four and two workloads respectively, and a time-slice value of 10ms allocated to the VM is divided equally among the workloads. The CPI values for labels 2.5ms and 5ms are normalized with respect to the values corresponding to 10ms. A large value on the Y-axis corresponds to a higher CPI and, therefore, the smaller the Y-axis value the better.

We show the behavior for all 29 SPEC CPU2006 benchmarks in Figure 3 to make our case. The benchmarks are sorted in ascending order of the performance degradation incurred as the allocated time-slice value is reduced. Throughout this paper, we identify the benchmarks in figures using the first four letters of their names. For applications that appear on the left of the figure, the CPI varies very little as the duration of the time-slice value is reduced from 10ms to 2.5ms. However, the CPI varies significantly in the case of applications that appear on the right. For the remaining applications, the variation in CPI as the time-slice value is decreased is distributed across the spectrum. The maximum degradation for a 2.5ms time slice is observed in case of HMMER and is 54%. An analysis of the results reveals that different applications indeed suffer from CS events differently; some suffer mildly while others suffer severely. Further, the CS performance penalty varies over the duration of execution of an application (section 5.1). The variation in performance degradation can be addressed by adopting ETS. The key insight behind ETS is to allocate fewer but longer time slices to address the performance penalty incurred by the most-affected workloads. To facilitate ETS, a dynamic mechanism is essential for estimating the extent to which an application suffers from CS events. We now describe an assumption and justify the reason for making it before presenting the dynamic mechanism.

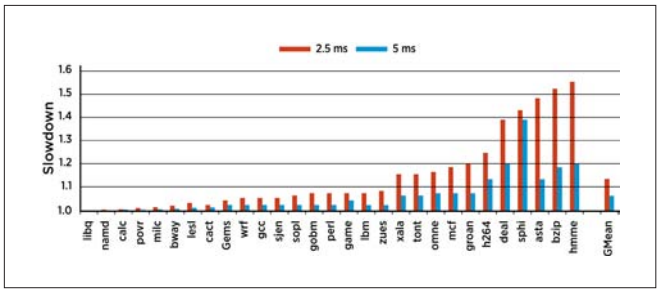


Figure 3. Variation in slowdown (measured in terms of CPI) as the time-slice value is reduced for SPEC CPU2006 benchmarks. The impact of CS events is significant on some workloads and imperceptible on others.

We assume that the data cached by an application in the LLC during the duration of its time slice is completely evicted by the data brought in by the intervening applications, before it is scheduled again. This assumption holds because of the aggressive multitasking employed by the virtualized systems described in section 1. We co-scheduled eight applications in a round-robin (RR) fashion, each for a time-slice duration of 2.5ms. This co-schedule is analogous to a scenario in which there are two VMs, each containing four workloads. The baseline time-slice value of 2.5ms is obtained by dividing 10ms equally among the workloads comprising a VM. The values we considered for the number of VMs and the number of workloads in this work are conservative. The actual numbers are even larger [1] [3], and our assumption is still valid under such conditions. We evaluated 30 different co-schedules, each made up of eight distinct applications, and observed the number of residual lines from one schedule of the application to its next schedule. Residual lines are those lines that remain in the cache from one schedule to the next. Over the total duration of execution, the number of residual lines for all applications is zero. Similar behavior was also observed in previous works [1] [3]. Even though the base time-slice value is small, when the execution of interleaved applications is considered, the total time is sufficient for an application's data in the LLC to be evicted completely. This phenomenon has two implications. First, invalidating the cache entries faithfully emulates a CS event. Second, there is no negative impact due to extending the time-slice value of an application on those applications whose time-slice value remains unchanged.

3. Framework for Estimating and Addressing the CS Performance Penalty

The motivational results presented in section 2 suggest that a dynamic mechanism is essential for estimating the penalty incurred due to a CS event. Such a mechanism can be used to characterize the impact of a CS event on an application. Now we describe the mechanism designed to estimate the cost of a CS event in terms of the number of CS misses incurred. The mechanism is capable of computing the cost of a CS event effectively while incurring a minimal overhead. Further, we present an augmentation to the baseline UTS RR CPU-scheduling algorithm in order to derive an ETS RR CPU-scheduling algorithm. The latter is capable of leveraging the calculated cost of CS events to mitigate the performance degradation.

3.1 Cost Estimation of a CS Event

The number of CS misses suffered by an application can be estimated in a simple but inefficient manner by making a copy of the tag directory (of the cache) on a CS event. When the application obtains a schedule again, the accesses that miss in the main tag directory but hit in the copy tag directory are tracked. The number of such accesses corresponds to the number of CS misses suffered. This simple scheme suffers from the following drawbacks. If there are multiple co-scheduled applications, we need a corresponding number of copy tag directories, which incur a significant area overhead. Multiple copy tag directories can be avoided by storing all but the one required (at any given time) in the memory. This approach requires maintaining space in the memory and logic to store and restore the copy tag directory to and from the memory respectively. Further, additional memory bandwidth is required to perform the store and restore operations. Now we propose a solution that overcomes these disadvantages. The solution is based on the following key ideas. CS miss count can be estimated by emulating a CS event. This requires only one copy tag directory (Figure 4a). The hardware overhead due to the copy tag directory can be reduced

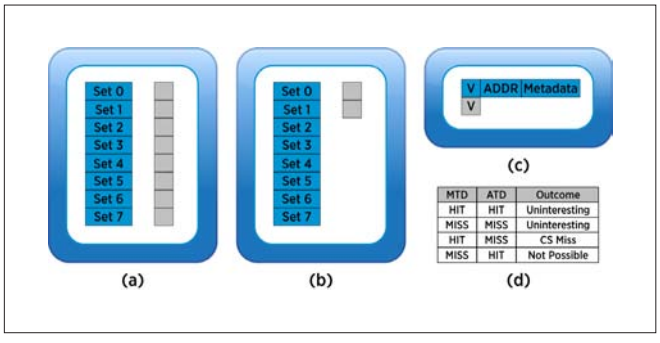


Figure 4. (a) The MTD (wide) and the full ATD (narrow) (b) The MTD (wide) and the ATD (narrow) with sample sets (c) An entry in the MTD (wide) and the ATD (narrow) (d) Status of accesses in the MTD and the ATD after a CS event.

by maintaining copy tags only for a fraction of the sets in the cache (Figure 4b). Further, the copy tag directory entry needs to contain only one bit of information (a valid bit), as opposed to a main tag directory entry that contains a valid bit, address bits, and other metadata (Figure 4c).

The working of our cost-estimation framework is modeled after that of a Monte Carlo (MC) method. MC methods rely on random sampling to determine an approximate answer to a question. In general, the answer determined using a MC method becomes more accurate as the number of samples considered increases. The proposed mechanism consists of an auxiliary tag directory (ATD) in addition to the regular main tag directory (MTD) in the cache. The ATD contains tags corresponding to a certain number of sets in the MTD. These sets in the MTD are referred to as *sample sets* (SS). We reason about the exact number of SS required in section 5. An entry in the ATD contains only one bit of information and can be either valid or invalid. It should be noted that a hit in the ATD is analogous to the line being valid and a miss to the line being invalid. At the start of execution, the state of SS in the MTD and the ATD is consistent, which means that lines in the MTD and the ATD are either both valid or both invalid.

To estimate the number of CS misses for an application, the entries in the ATD are invalidated. This process emulates a CS event. After the point of invalidation, corresponding to subsequent cache accesses, one of the following scenarios can arise (Figure 4d): access hits in both the MTD and the ATD, access misses in both the MTD and the ATD, or access hits in the MTD but misses in the ATD. The first two events are not of interest to us. The third event corresponds to a CS miss. A miss in the ATD and a hit in the MTD happen because the ATD experienced a cache-flush event, which is analogous to a CS event. The corresponding entry in the ATD is made valid on recording the CS miss. So, further accesses to the same cache line do not generate CS misses. The ATD entry corresponding to the second event is made valid as well. We use a counter (CS-MISS-CNT) to keep track of the CS misses. The counter value is read at the time when the application is being switched out. It indicates the number of CS misses suffered by SS. To estimate the total number of CS misses experienced by the application, the counter value is multiplied with the ratio of the total number of sets to the number of SS. This ratio is chosen to be a power of 2 so that the multiplication operation degenerates to a simple left-shift operation. After the invalidation point, an event corresponding to a miss in the MTD and a hit in the ATD does not happen by construction (Figure 4d). The set of hits in the ATD is always a proper subset of the set of hits in the MTD. We depict the steps for estimating the CS miss count in Algorithm 1, using pseudocode.

The mechanism proposed above aids in estimating the number of CS misses for a private cache. The trend in modern computer systems is to employ simultaneous multiple threading (SMT) and/or multiple cores to enhance performance while keeping power consumption in check. We refer to the hardware thread instances in the case of SMT and the cores in a multicore processor commonly as *sharers*. When the cache is shared by two or more sharers, the CS cost-estimation mechanism needs to be augmented as follows to support the cost estimation for each sharer. The modification

```
initialize() {
 invalidate entries in ATD; CS-MISS-CNT = 0; }

count_cs_misses() {
 if ((MTD.lookup == hit) &&
 (ATD.lookup == miss)) CS-MISS-CNT++; }

estimate_cs_penalty() {
 CS-MISS-CNT X (number-of-sets-in-cache/
 number-of-sample-sets); }
```

Algorithm 1. CS cost computation in terms of the number of CS misses

required is to replicate CS-MISS-CNT counter per sharer. Because lines belonging to each sharer are uniquely identified in the tag entry of the cache, the identifier can be used to match and update the corresponding counter. Note that one copy of the ATD is sufficient irrespective of the number of sharers.

3.2 Design of an ETS RR CPU-Scheduling Algorithm

Previously, we pointed out that fewer but longer time slices must be used for those applications that are severely impacted by CS events. By doing so, we can alleviate the negative impact of CS events on the performance of such applications. In this section, we describe the design of a CPU-scheduling algorithm that achieves this goal. Specifically, we augment the baseline UTS RR CPU-scheduling algorithm to derive an ETS RR CPU-scheduling algorithm. Recall that we described the distinction between the two in section 1. To keep the discussion precise, we choose the following values for parameters (same as the values used throughout this paper). The baseline UTS algorithm allocates a time-slice value of 2.5ms for all applications in a RR fashion. We consider a system with a LLC capacity of 2MB. The cache consists of a total of 32,768 lines, each of size 64 bytes. The ETS algorithm categorizes these lines into four groups, as shown in the Group column of Table 1. The groups are based on the number of CS misses. For an application belonging to a particular group, the algorithm extends the time slice to the value indicated in the Slice column. The size of the group is doubled from one group to the next, and the time-slice value is increased by 2.5ms. In this manifestation, we capped the maximum time-slice value at 10ms. In an actual system, we expect that this value will be set after taking the response-time constraints and other factors into consideration.

	GROUP	SLICE		GROUP	SLICE
(1)	≤1,500	2.5 ms	(2)	1,501 - 4,500	5.0 ms
(4)	>10,501	10 ms	(3)	4,501 - 10,500	7.5 ms

Table 1. An ETS round-robin CPU-scheduling algorithm implementation. CS miss count is used as index in order to determine the time slice value for the next schedule.

We measure the number of CS misses experienced by the application every time it obtains a schedule on the processor. The number of misses estimated using Algorithm 1 is used as index into Table I. The corresponding value of Slice is assigned as the time-slice value for the next schedule of the application. It must be noted that the discretization presented in Table 1 is realized in software and therefore can be customized to a target system. We developed the presented

discretization by heuristically running the benchmarks and analyzing the results. A high-level overview of the working of the ETS framework is provided as a flow chart in Figure 5.

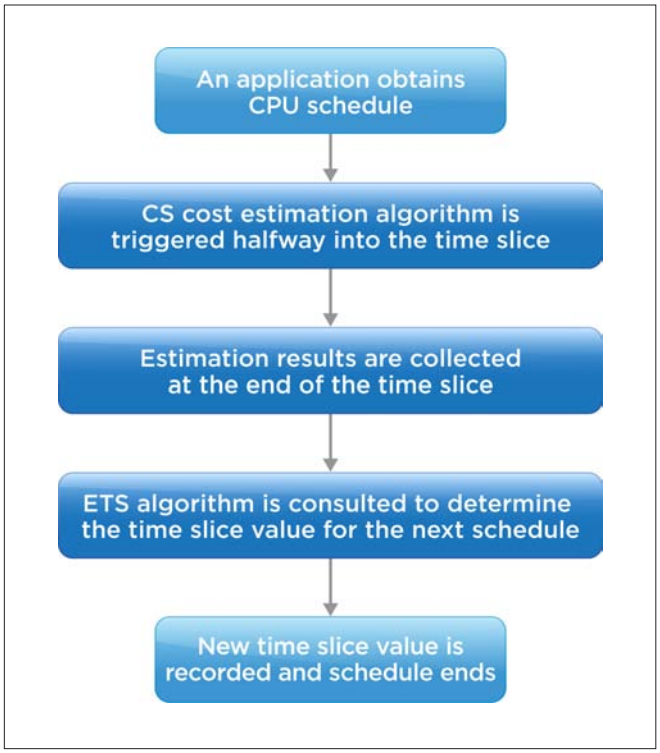


Figure 5. High-level overview of the ETS framework

It is not our objective to propose an alternative CPU-scheduling algorithm. There is a large body of work that investigated such algorithms. However, we argue that these algorithms must be supplemented to make them aware of the cost of the CS events. The design of the scheduling algorithm could incorporate CS miss information together with other currently used factors such as priority and interactivity. Here, we described how a UTS RR CPU-scheduling algorithm can be augmented to account for the CS penalty incurred.

4. Experimental Methodology

We use an in-house trace-driven simulator to conduct the experiments. The processor is modeled as an in-order core, and the simulator is capable of handling multiple cores. The memory hierarchy consists of three levels of caches: separate instruction and data caches at the first level, and unified caches at the middle and the last levels. A uniform value of 64 bytes is used for line size across the entire hierarchy. We use a baseline value of 2MB for the LLC capacity in our experiments. Our simulator can model the LLC as private to each core or as shared among multiple cores. In either case, multiple applications can be co-scheduled on each core. All cache levels implement the LRU replacement policy. The parameters of the simulated machine are shown in Table 2.

Processor	4GHz, Single Issue, In-Order
L1 I-cache	32KB, 2-way
L1 D-cache	32KB, 2-way
L2 cache	256KB, 4-way
LLC	2MB, 16-way, 24 cycles
Main memory	400 cycles

Table 2. Machine configuration

In the event of a context switch, the employed framework eliminates effects other than the loss of saved state in the LLC. We use all benchmarks (29 in number) from the SPEC CPU2006 suite to obtain a comprehensive set of results. Each benchmark is comprised of a representative set of 500 million instructions. For our experiments, we combined disparate benchmarks to generate 29 diverse workload mixes (co-schedules). When an LLC capacity other than 2MB is used, we keep the associativity constant and increase the number of sets. We apply the CS cost-estimation mechanism to the LLC in the system as the distance between the LLC, and the main memory is far in units of CPU cycles. Our baseline system employs the UTS RR CPU-scheduling algorithm and uses 2.5ms for the time-slice value. The UTS algorithm is representative of the mechanism in IBM PowerVM virtualization, in which a fixed scheduling period can be shared by up to 10 vCPUs through micropartitioning. The prominent parameters used in this work are modeled after those used in the most recent related paper [3]. These include the number of co-scheduled applications, the baseline CPU-scheduling algorithm and time-slice value, and the capacity of the LLC. Further, the LLC capacity of 2MB per core used in this work is representative of the LLC capacity in server class machines.

5. Results and Analysis

Hereafter, we use the word *cache* to refer to the LLC by default. We now attempt to answer the following questions by relying on experimental results: What is the performance improvement that can be obtained by adopting ETS? How accurately can we estimate the number of CS misses?

5.1 Advantage of Using the ETS CPU-Scheduling Algorithm

The results obtained by employing the ETS RR scheduling algorithm described in section 3.2 are shown in Figure 6 for all SPEC CPU2006 benchmarks. The results correspond to a cache capacity of 2MB. In each case, a total of eight applications are co-scheduled onto a single core. We study the impact of CS events on the application indicated by the X-axis label, which is the application of interest. We apply the ETS algorithm to it and modify its time-slice value. The time-slice value for the remaining applications is 2.5ms, which is the baseline time-slice value of the UTS RR CPU-scheduling algorithm. The evaluation metric used is the IPC corresponding to the execution of 500 million instructions. The values corresponding to the ETS algorithm are normalized with respect to the values for the UTS algorithm.

Figure 6(a) shows the improvement in IPC obtained by employing the ETS algorithm compared to that obtained using the UTS algorithm (2.5ms time slices). The applications are sorted in ascending order of the benefit derived from the ETS algorithm. Figure 6(b) shows the distribution of time slices allocated by the ETS algorithm. Some applications, such as libquantum and CalculiX, are minimally impacted by CS misses. The time-slice allocation distribution shows that the time slices allocated to these applications are predominantly of duration 2.5ms. In contrast, applications such as HMMER, bzip2, and astar are severely impacted by CS misses. The distribution shows that the time slices allocated to these applications are predominantly of duration 5ms, 7.5ms, and 10ms. The ETS algorithm allocates longer time slices on the basis of their utility to applications. The maximum improvement in IPC is obtained in case of HMMER and is as much as 54%. The remaining applications span the entire spectrum of performance improvement.

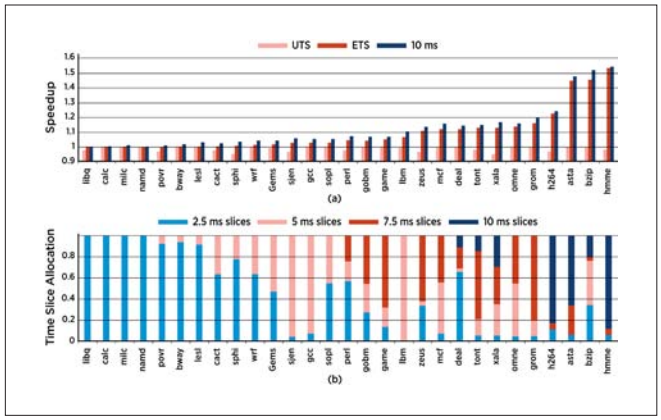


Figure 6. (a) IPC improvement by adopting the ETS RR CPU-scheduling algorithm (b) Distribution of allocated time slices

The diversity of the results shown in Figure 6 reinforces our hypothesis that we should track the number of CS misses dynamically and allocate longer time slices to those applications that suffer from CS misses significantly. Out of a total of 29 applications studied, the performance improvement due to the ETS algorithm is 5% or more in the case of 15 applications and more than 10% in the case of 11 applications. Figure 6(a) also shows the IPC results corresponding to the case when a constant value of 10ms is used for the time slice. The results are once again normalized with respect to those for the UTS algorithm (2.5ms time slices). The IPC results obtained using the ETS algorithm are within 4% of the results obtained using a constant value of 10ms for the time slice. In summary, the results provide substantial evidence in favor of the ETS algorithm to address the negative performance impact of CS events. It should be noted that the ETS algorithm is implemented in software. Therefore, it can be customized and optimized for a target system. However, we expect that the implementation will be kept simple to contain the direct overhead associated with a CS event. In our implementation, the additional cost is approximately 10 instructions.

The cumulative CPU time allocated by the ETS algorithm to all applications (including the application of interest) is equal to that allocated by the UTS algorithm. We demonstrate this using an example in Figure 7. For clarity of discussion, we consider the case in which there are a total of three co-scheduled applications. However, the reasoning also applies to co-schedules involving a different number of applications. In Figure 7, P3 is the application of interest. The time-slice allocation performed by the UTS algorithm is shown in the row labeled 2.5ms. The time-slice allocations made by the ETS algorithm for three different scenarios are shown in the other rows. Although the ETS algorithm allocates longer time slices, it allocates fewer such longer time slices. As the length of the allocated time slice increases, the number of allocations of the time slice decreases. Therefore, the ETS algorithm offers the same fairness guarantees as the UTS algorithm.

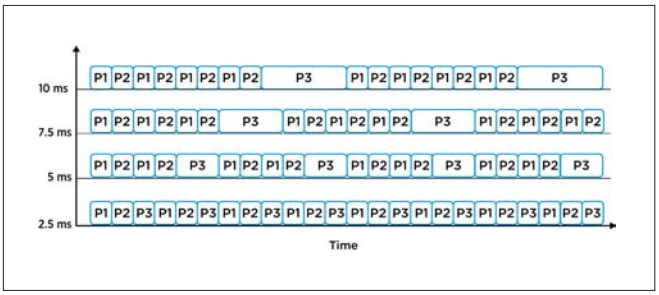


Figure 7. Example to illustrate the allocation of time slices by the ETS algorithm

Latency or response time—time elapsed between two consecutive schedules of an application—is another important aspect of a CPU-scheduling algorithm. In Figure 8, we provide quantitative information regarding the latency behavior of three algorithms: UTS algorithm with 10ms time slices, UTS algorithm with 2.5ms time slices, and ETS algorithm. For an application indicated by the X-axis label, the Y-axis value corresponds to the latency incurred by a co-scheduled application. The average latency for UTS-10, UTS-2.5, and ETS is 30ms (horizontal solid line), 7.5ms (horizontal dotted line), and 7.5ms (rectangles) respectively. Whereas the standard deviation in latency for UTS-10 and UTS-2.5 is 0, the value for each application in the case of ETS is shown in the form of error bars above the rectangles. The maximum value of the standard deviation is 3.7 and is observed in case of HMMER. From the data presented in Figure 8, it can be inferred that the latency behavior of the ETS algorithm is very similar to that of UTS-2.5. In addition, the performance behavior of the ETS algorithm is nearly identical to that of UTS-10 (Figure 6(a)). The ETS algorithm adapts to the dynamic behavior of the applications to achieve the best of both worlds.

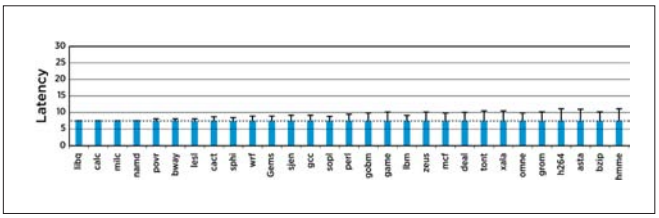


Figure 8. Latency behavior of UTS-10 (horizontal solid line), UTS-2.5 (horizontal dotted line), and ETS algorithms. Error bars indicate the standard deviation in latency for the ETS algorithm. The latency behavior of the ETS algorithm is very similar to that of UTS-2.5.

The CS performance penalty varies not only across applications but also over the duration of execution of an application. This is because applications go through phases of execution. We provide

experimental results in Figure 9 to substantiate our claim. Figure 9 shows the time-slice transitions for all benchmarks over their total duration of execution. The label **Same** indicates the fraction of transitions from a time-slice value to the same time-slice value, and the label **Different** indicates the fraction of transitions to a different time-slice value. A **Different** transition happens when the number of CS misses changes considerably from a time slice to the next. Hence, a large value for the fraction of **Different** transitions is indicative of the change in the CS miss behavior over the duration of execution. The fraction of **Different** transitions is 10% or more for a total of 19 applications. A maximum value of 68% is recorded in case of xalancbmk. The results corroborate our hypothesis that the CS miss behavior indeed varies over the duration of execution of an application.

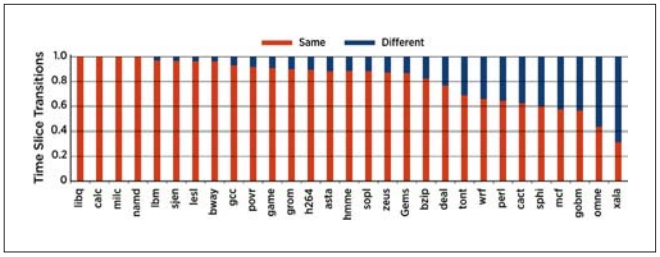


Figure 9. Variation in CS miss behavior over the duration of execution of applications. Same label indicates the fraction of transitions from a time-slice value to the same time-slice value, and Different indicates the fraction of transitions to a different time-slice value. A large value for the fraction of Different transitions indicates that the CS miss behavior changes over the duration of execution.

5.2 CS Cost-Estimation Accuracy

The ability to dynamically estimate the cost of a CS event is central to the operation of the ETS algorithm. We use the augmented CS cost-estimation mechanism described in section 3.1 to estimate the number of CS misses per sharer. The corresponding results are presented in Figure 10. Specifically, we provide the results when two cores share a 4MB cache. The sharers are identified uniquely through X-axis labels. The experiment used 256 SS, which correspond to 1/16 of the total number of sets. We evaluated various values for the number of SS and narrowed it down to 1/16 of the total number of sets. This choice achieves a good trade-off between area and accuracy. The estimation error is represented in percentage terms and indicates the separation between the value computed using the SS and the actual value. The average value of the estimated error across all sharers is 2.5% (excluding milc). The average error and the estimated error for most applications are both below 5%, indicating the usefulness of the proposed mechanism. We address the inaccuracy in estimation for milc in section 5.3.

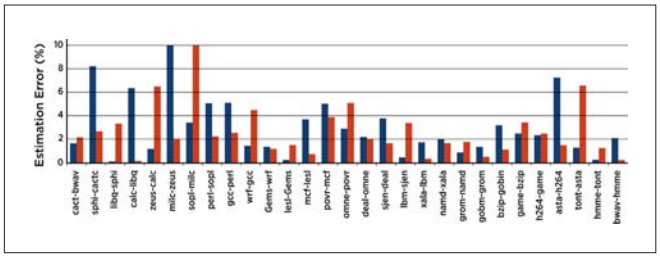


Figure 10. Percentage error in estimation of the number of CS misses when two applications share a 4 MB LLC

It is important to consider the mechanism that is employed to partition the cache among the sharers and how the mechanism affects CS miss count estimation. The results presented in this section are for the case when the ways are equally partitioned among the sharers and the LRU replacement policy is employed within each partition. We will now discuss the impact of more-advanced partitioning mechanisms on the accuracy of CS cost estimation. Global LRU replacement policy allows for dynamic sharing based on demand. However, it was previously shown that demand for cache does not always translate to benefit from cache [4]. Several proposals were made to improve the benefit derived from a shared cache: utility-based cache partitioning (UCP) [4], thread-aware dynamic insertion policy (TADIP) [5], and software-based shared-cache management techniques such as page coloring. The common goal of these works is to determine what is likely to be the optimum partition of the shared cache and enforce the applications to stay within the limit of the determined optimum partition. These methods allocate ways of sets or sets of cache among the applications. Such structured allocation lends itself well to the proposed CS cost-estimation mechanism, which works on the principle of uniform sampling. In summary, we anticipate that employing the proposed CS cost-estimation scheme in conjunction with advanced partitioning mechanisms will result in as accurate estimates as we obtained here.

We also evaluated the accuracy of the CS cost-estimation mechanism for a 2MB private cache and when four cores share an 8MB cache. The average value of the estimated error across all workloads is 2.7% and 2.5% respectively (excluding milc). The results corresponding to the private cache are shown in Figure 11. Estimating the number of CS misses to within a 10% value can provide very important information for a CPU-scheduling algorithm to factor the CS event cost. More concretely, we anticipate that the trend for the number of CS misses rather than the actual value will be used by the CPU scheduler. We discussed the performance improvement obtained using one manifestation of such a scheduler in section 5.1.

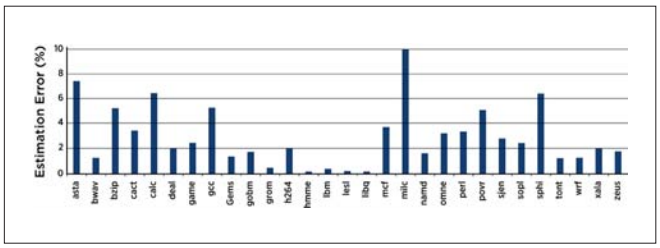


Figure 11. Percentage error in estimation of the number of CS misses for a 2MB private LLC

5.3 Addressing CS Cost-Estimation Inaccuracy

In section 5.2, we pointed out that the estimated value of CS misses is inaccurate (by 50%) for milc. We now propose a mechanism, which incurs minimal overhead, to determine when the CS miss count estimate is inaccurate. The ATD presented in section 4.1 is organized as two logical entities, with each one comprising half the original number of sample sets. We associate each logical entity with a separate CS-MISS-CNT counter. The hardware overhead of the enhanced estimator is this additional counter and a small amount

of logic necessary to organize the logical entities. It should be noted that the physical organization of the ATD is still the same as it is for the original estimation mechanism described in section 3.1.

The estimation mechanism works as before, with the SS updating the respective counters for the number of CS misses. When the application is being switched out, the values of the two counters are used to determine if the estimated value is accurate. Specifically, we compute the ratio of the absolute difference of the two counter values and their sum. A large value of this ratio indicates that the estimate is inaccurate. Otherwise, the computed sum of the two counter values serves as the estimated value for the number of CS misses corresponding to the SS. This number is scaled to the total number of sets in the cache to get the final estimated value. The specified ratio is provided for all benchmarks in Figure 12. The value for milc is 0.30, while the corresponding value for all other benchmarks is at most 0.15. The computed ratio is a measure of divergence between the two counter values. When the estimate is inaccurate, the divergence is large. If this is the case, we ignore the estimate and keep the time-slice value intact.

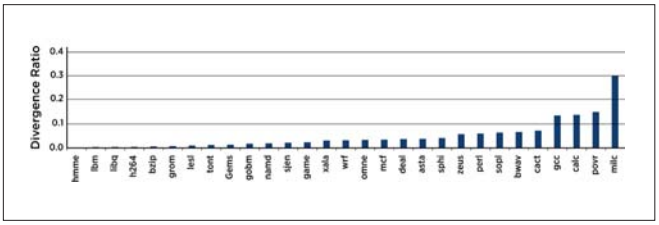


Figure 12. Divergence ratio of CS-MISS-CNT counter values, used to determine the accuracy of CS cost estimate. A large value of the ratio denotes that the CS cost estimate is inaccurate.

5.4 Hardware Overhead

In this section, we quantify the additional hardware necessary to support the ETS mechanism. The storage component of the overhead consists of the following: (1) tags for the sample sets in the ATD and (2) counters for tracking the number of CS misses. This component is computed in Table 3. We assume a physical address space of 40 bits and use the same baseline LLC capacity of 2MB used previously. The LLC is organized as a 16-way associative cache for a total of 2,048 sets. Note that we used 1/16 of total sets for the number of SS throughout this paper. Additional logic is required in order to invalidate the ATD entries (to emulate a CS event), detect a CS miss (a 2-ip logic gate), and increment the CS- MISS-CNT counter. The overhead due to the extra logic is negligible, similar to the storage overhead.

Size of an MTD entry (address + metadata)	4B
Size of an ATD entry (valid-bit)	1 bit
Number of ATD entries (16 per set × 128 sets)	2048
Overhead of ATD	256B
Size of counter *(log 2 {number – of – cache – lines – in – ATD})*	1.5B
Overhead due to 2 counters	3B
Area of baseline LLC (128kB tags + 2MB data)	2176kB
% increase in LLC area (260B/2176kB)	0.012%

Table 3. Storage overhead to support the ETS mechanism

Prior to developing the sampling-based CS cost-estimation framework, we attempted to make use of the change in the IPC of the task before and after a CS event to estimate the cost of the CS event. The advantage of this approach is that it does not require any additional hardware. It makes use of the hardware-performance counters built into modern microprocessors. However, this mechanism suffers from the following drawbacks. First, in several instances, the IPC value before and after a point where a CS event will be scheduled are different. This inherent change in the IPC over the duration of execution serves to increase or decrease the actual cost of a CS event. Second, the duration for which the IPC value must be tracked varies across instances. In our experiments, after a CS event, we noticed that the CS misses are interspersed between regular misses, and the distribution of CS misses within the regular misses varies. This spread of CS misses adds to the problem of inherent difference in the IPC value. We developed the sampling-based framework after realizing the limitations of the IPC-based mechanism.

6. Impact of Cache Optimizations on CS Misses

In Section 1, we mentioned that the mechanisms that attempt to improve the cache hit rate by addressing the traditional cache misses exacerbate the problem associated with CS misses. Such optimizations attempt to retain more data that will be useful in the future, and CS events result in the loss of this data. Now, we evaluate the impact of improving the replacement algorithm and increasing the capacity of the cache on the number of CS misses. We also assess the accuracy of the CS cost-estimation hardware when it is applied to advanced replacement algorithms. Furthermore, we provide the performance-improvement results obtained by using the ETS mechanism in conjunction with these algorithms. In our experiments, the first-level and the middle-level caches use the LRU replacement policy, and our replacement-policy studies are limited to the LLC.

6.1 Replacement Algorithm

Thus far, we have assumed that the cache implemented the LRU replacement policy. It performs poorly in the following two scenarios: when the size of the working set is larger than the capacity of the cache, and when references to nontemporal data (scans) cause a frequently referenced working set to be discarded. Solutions were proposed to address one or more of these shortcomings of the LRU algorithm: dynamic insertion policy (DIP), rereference interval prediction (RRIP), and signature-based hit prediction (SHiP). DIP chooses between two policies—bimodal insertion policy (BIP) and LRU policy—dynamically depending on which policy incurs fewer misses. The selection is made through dynamic set sampling and set dueling [6]. RRIP policy works by predicting the rereference interval of a cache line. The work proposes two policies: static RRIP (SRRIP) and dynamic RRIP (DRRIP). DRRIP again uses set dueling to identify which of SRRIP and bimodal RRIP (BRRIP) performs the best [7]. Finally, SHiP works by correlating the rereference behavior of a cache line with its signature [8]. We use the program counter (PC) value to derive the signature.

Now, we present the results to show how the estimation hardware performs for replacement policies other than LRU. Specifically, we present the results for DRRIP and SHiP algorithms. The results

correspond to a private cache with a capacity of 2MB. In Figure 13(a), we show the percentage error in estimation of the number of CS misses. For each algorithm, we present the results generated using 128 SS. Using 128 SS, which correspond to a 1/16 fraction of the total number of sets, we obtain an accurate estimate for the number of CS misses. The average percentage error in estimation is 2.8% and 2.4% for DRRIP and SHiP algorithms (excluding milc) respectively. The estimation error for milc is 29% and 18% respectively. These results demonstrate that the estimation hardware, because it is based on sampling, lends itself very well to other replacement algorithms.

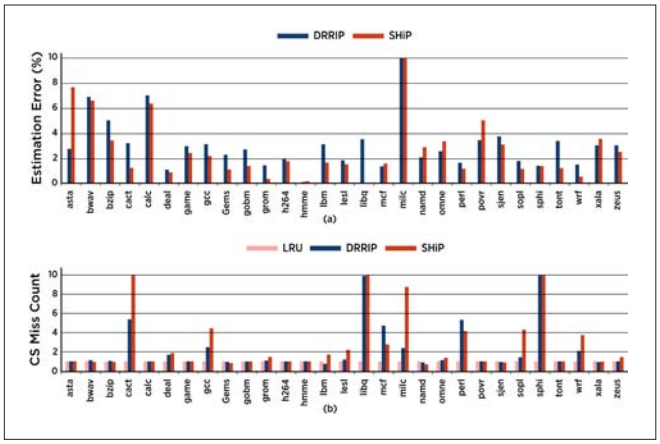


Figure 13. (a) Percentage error in estimation of the number of CS misses (b) Impact of advanced replacement algorithms on the number of CS misses

Next, we consider the impact of the replacement algorithm on the number of CS misses suffered by an application. In Figure 13(b), we show how the number of CS misses varies with the replacement algorithm. The values for DRRIP and SHiP are normalized with respect to the values for LRU. For several benchmarks, the number of CS misses increases pronouncedly for DRRIP and SHiP when compared to LRU. The maximum increase in the number of CS misses (by a factor of 20X) is observed in case of sphinx3 for both DRRIP and SHiP. The geometric mean across all benchmarks is 1.6 and 2 for DRRIP and SHiP respectively. These results substantiate our hypothesis that adopting advanced replacement algorithms accentuates the problem associated with CS misses.

The IPC results obtained by employing the ETS RR scheduling algorithm are shown in Figures 14(a) and 14(b) for DRRIP and SHiP policies respectively. The experimental methodology used is similar to that employed in section 5.1. The values corresponding to the ETS algorithm are normalized with respect to the values for the UTS algorithm (2.5ms time slices). The applications are sorted in ascending order of the benefit derived from the ETS algorithm. Figure 14 also shows the IPC results corresponding to a 10ms time-slice value normalized with respect to the results for the UTS algorithm. The IPC results obtained using the ETS algorithm are within 4% of the results obtained using a constant value of 10ms for the time slice. It can be inferred from these results that the ETS algorithm is equally applicable for advanced replacement policies. For both DRRIP and SHiP policies, the performance improvement due to the ETS algorithm is 10% or more in case of 11 applications.

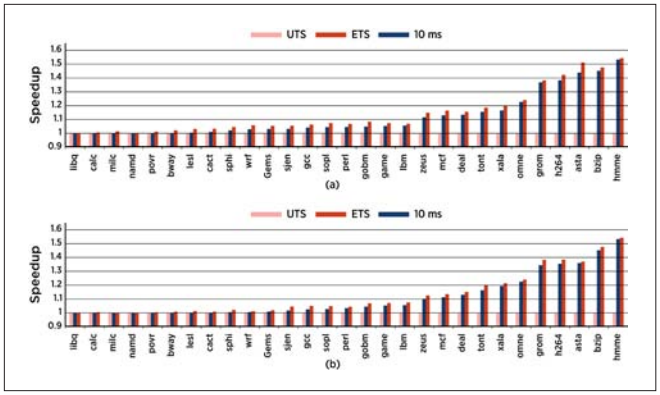


Figure 14. IPC improvement by adopting the ETS RR CPU-scheduling algorithm for (a) DRRIP and (b) SHiP replacement algorithms

6.2 Cache Size

We now consider the impact of another important optimization—increasing the capacity of the cache—on the number of CS misses. We present the results for all the benchmarks in Figure 15. We obtained the results for three different cache sizes: 1MB, 2MB, and 4MB. We provide the evaluation results for three replacement algorithms: LRU, DRRIP, and SHiP.

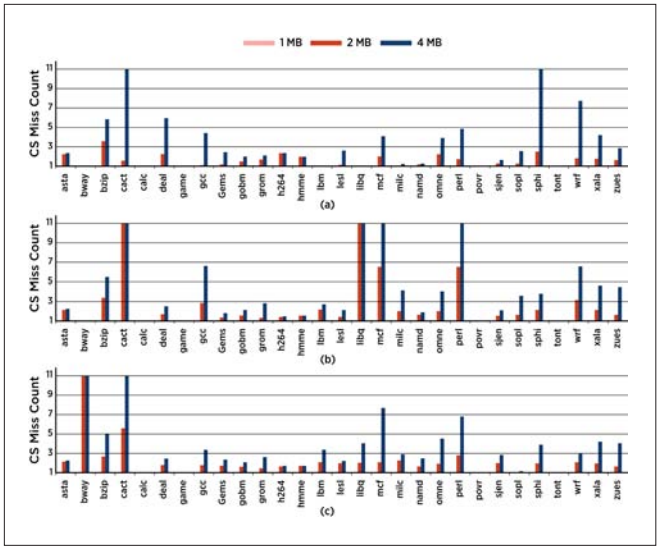


Figure 15. Impact of increasing the cache size on the number of CS misses for (a) LRU, (b) DRRIP, and (c) SHiP replacement algorithms

The number of CS misses for 2MB and 4MB cache sizes are normalized with respect to that for the capacity of 1MB. In general, for all benchmarks and replacement policies, the number of CS misses increases as the cache size increases. In several cases, the increase is by a factor of 5X or more from 1MB capacity to 4MB capacity. For cactusADM, with the DRRIP replacement policy, the number of CS misses increases by a factor of 101X from 1MB capacity to 4MB capacity. In summary, the results indicate that increasing the cache capacity has a significant impact on the number of CS misses experienced by the application. The performance penalty due to the increased number of CS misses will be commensurate with the magnitude of increase in the number.

In summary, cache optimizations accentuate the problem associated with CS misses. Therefore, in the presence of such optimizations, estimating the CS miss cost accurately and incorporating the estimate into CPU-scheduling algorithms become even more important.

7. Related Work

Studies related to context switching have received much attention from both the industry and the academia over a long period of time. We summarize, compare, and contrast the works that closely relate to the techniques proposed in this paper under four different categories.

7.1 Performance Impact of Context Switching

Many studies aimed at understanding the performance impact of CS events. Agarwal et al. [9] showed that multiprogramming activity significantly degrades cache performance, and that the impact grows with increase in cache size. Mogul et al. [10] estimated the performance reduction caused by a CS to be in the order of tens to hundreds of microseconds, depending on the cache parameters. Suh et al. [11] considered the performance impact of context switching on page faults. They proposed to mitigate the problem using job speculative prefetching. Chiou et al. [12] proposed that memory scheduling, potentially at all levels of the memory hierarchy, should drive CPU scheduling rather than the other way around as it is done in most systems.

Koka et al. [13] characterized CS misses and quantified their impact in the case of transactional workloads. They investigated the potential for intelligent process scheduling that minimizes cache misses across CS boundaries. Li et al. [14] concluded that the indirect CS overhead due to cache perturbation is more significant than the direct overhead. Tsafrir [15] and David et al. [16] calculated the indirect overhead due to a CS event for Intel and ARM platforms respectively. In summary, most of the works concluded that the indirect overhead, due to cache perturbation, associated with CS events is significant. It is this overhead that we attempt to address through our proposal.

7.2 Analytical Models

Several analytical models were proposed to explain the relationship between an application's temporal reuse behavior and its vulnerability to CS misses. Such models need to factor in all essential variables to have a sufficient resolution. Agarwal et al. [17] and Suh et al. [18] [19] proposed analytical models to estimate the overall cache miss rate, including fully associative cache, in order to obtain a continuous miss-rate curve, which is required as profiling information. This assumption does not hold true in case of the LLC. The model by Hwu et al. [20] was aimed at predicting the worst-case number of CS misses. Liu et al. [21] classified CS misses into two categories: replaced misses and reordered misses. Further, they developed an analytical model to reveal the relationship among cache design parameters, an application's temporal reuse pattern, and the number of CS misses the application suffers from. They applied the devised model to study the impact of prefetching and cache size on the number of CS misses.

Analytical models make certain assumptions to render the task of making the model tractable. For example, the model by Liu et al. [21] is designed under the assumption of LRU replacement policy. However, advanced replacement algorithms were proposed that perform better than the LRU algorithm. Also, analytical models are suitable for offline analysis, but the feasibility of their implementation in hardware while incurring a low area overhead is not considered. Our solution's approach is implemented using very low hardware overhead to work in a dynamic environment for any cache configuration, thereby addressing the previously pointed-out limitations of the analytical models.

7.3 Employing Prefetching to Cope with CS Misses

The performance degradation due to CS misses can be addressed through two different means: by (1) increasing the time slice value (2) prefetching the cache state just before or when the new schedule starts. The former method is a preventive measure and the latter is a cure. Prefetching was suggested to mitigate the cost of additional cache misses incurred because of a CS event. The general idea is to record the application's locality at the time when it gets swapped out. The locality is restored through prefetching the next time application gets CPU time. Previously proposed solutions that employ prefetching differ in how the locality is stored and restored. Cui et al. [22] employ global-history-list (GHL) prefetching. GHL maintains a complete list of cache lines, which is ordered by recency of use. Daly et al. [1] studied the impact of CS misses in highly partitioned virtualized systems. They proposed cache restoration as a solution to prefetch the working set and thereby warm the cache. GHL and cache restoration, although they differ in implementation details to some extent, perform similarly. GHL performs slightly better at the expense of more hardware and complexity. In the most recent related work [3], the authors proposed methods to reduce the bandwidth overhead of these prefetchers.

Brown et al. [23] proposed accelerating postmigration thread performance by predicting and prefetching the working set of the application. In the proposed solution, access behavior of a thread is captured and summarized into a compact form premigration. On the new core, the summary is used to prefetch appropriate data to create a warm state. Prefetching the data after a CS event serves to cure the cold-start problem. However, ETS works to minimize the number of cold starts for those applications for which it matters. The techniques presented in this paper can provide guidance as to when prefetching can be beneficial and when it is not likely to help. Zebchuk et al. [3] identified the inability of all cache-restoration prefetchers to dynamically adapt to the workload behavior as their main limitation. Our framework can be potentially used in conjunction with prefetching to address this key drawback. They can complement each other to achieve a synergistic effect.

7.4 Dynamic Set Sampling

Dynamic set sampling (DSS) was previously used to achieve multiple goals. The key intuition behind set sampling is that it is sufficient to monitor a relatively small fraction of the sets in the cache in order to understand the behavior of the entire cache. DSS was used in conjunction with set dueling to decide which of two or more policies performs the best at any given point. This technique was used to select the best-performing replacement policy: LIP versus BIP [6], MLP-aware versus traditional [24], and SRRIP versus BRRIP [7]. In a system with private LLCs, it was also used to determine if each cache should act as a spiller or a receiver [25]. In the context

of a shared cache, it was used to determine whether each thread among a group of threads sharing the cache should implement LIP versus BIP policy [5]. Also, in the context of a shared cache, DSS was used independently (without set dueling) to partition the ways of the cache in the best possible manner by monitoring utility [4]. To our knowledge, this is the only instance in which DSS is used to estimate the absolute value of a parameter as we used it to estimate the number of CS misses.

8. Conclusion

In a system employing multitasking, an application suffers from cache misses due to CS events in addition to the typical cache misses. CS misses happen as a result of the displacement of the cache state, which is caused by other applications intervening between two consecutive schedules of an application of interest. CS misses are more of a problem in systems that support multitasked virtualization. Such systems experience severe cache pollution as a consequence of the additional degree of multitasking, above and beyond the regular OS-level multitasking. However, the extent to which an application suffers from CS misses varies from one to another, depending on the temporal reuse behavior. Whereas some applications suffer only mildly, others suffer severely. We made the following contributions through this paper:

- We demonstrated that applications suffer by varying degrees because of context switching. In response to this phenomenon, we proposed to estimate the penalty due to a CS event and use it to facilitate intelligent time slicing by employing ETS. The intuition behind ETS is to provide longer yet infrequent time slices to those applications that are affected severely, while keeping the time slices allocated to the unaffected applications intact.
- We developed a hardware-based dynamic CS cost-estimation mechanism that incurs low area overhead. We characterized the accuracy of estimation of the proposed mechanism for multiple configurations and showed that the mechanism is very reliable.
- We provided insights into how the CS cost estimate can be incorporated into the design of a CPU-scheduling algorithm. We validated the potential of ETS to reduce the negative impact of CS events on performance without sacrificing response-time behavior.
- Furthermore, we evaluated the impact of advanced replacement algorithms and increasing the cache size on CS misses and found that these optimizations aggravate the problem associated with CS misses.

The ETS algorithm developed in this paper allocates longer time slices on the basis of their utility to applications. For various cache-management policies, the speedup obtained using the ETS algorithm is within 4% of that realized using a constant value of 10ms for the time slice. We augmented the UTS RR CPU-scheduling algorithm in order to derive the ETS RR CPU-scheduling algorithm. However, the ETS algorithm is implemented in software and can be optimized for a target system. One possible direction for future research is to investigate how CS cost estimate can be incorporated into other CPU-scheduling algorithms while respecting their original objectives. The hardware overhead of the proposed CS cost-estimation mechanism is only 0.01% for a 2MB cache. We used the estimated cost of a CS event, in terms of the number of CS misses, to modify the time slice in an elastic manner. In the case of cache-restoration prefetchers, the estimated number of CS misses can provide guidance as to when prefetching can be beneficial and when it is not likely to help. The inability of all cache-restoration prefetchers to dynamically adapt to the workload behavior has been identified as their main limitation. Our CS cost-estimation framework can be potentially used in conjunction with them to address the specified key drawback.

Acknowledgments

This work was supported in part by the Ministry of Science, ICT & Future Planning, Korea, under the R&D program supervised by the Korea Communications Agency (KCA- 2013-11921-03001), VMware, and C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

References

1. D. Daly and H. W. Cain, “Cache restoration for highly partitioned virtualized systems,” in *International Conference on High Performance Computer Architecture (HPCA)*, 2012, pp. 1–10.
2. A.-C. Lai, C. Fide, and B. Falsafi, “Dead-block prediction & dead-block correlating prefetchers,” in *ISCA*, 2001, pp. 144–154.
3. J. Zebchuk, H. W. Cain, V. Srinivasan, and A. Moshovos, “Recap: a region-based cure for the common cold cache,” in *International Conference on High Performance Computer Architecture (HPCA)*, 2013, pp. 83–94.
4. M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in MICRO, 2006, pp. 423–432.
5. A. Jaleel *et al*., “Adaptive insertion policies for managing shared caches,” in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 208–219.
6. M. K. Qureshi et al., “Adaptive insertion policies for high performance caching,” in *International Symposium on Computer Architecture (ISCA)*, 2007, pp. 381–391.
7. A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, “High performance cache replacement using re-reference interval prediction (RRIP),” in *International Symposium on Computer Architecture (ISCA)*, 2010, pp. 60–71.
8. C.-J. Wu *et al*., “Ship: signature-based hit predictor for high performance caching,” in *MICRO*, 2011, pp. 430–441.
9. A. Agarwal, J. Hennessy, and M. Horowitz, “Cache performance of operating system and multiprogramming workloads,” *ACM Trans. Comput. Syst*., vol. 6, no. 4, pp. 393–431, 1988.
10. J. C. Mogul and A. Borg, “The effect of context switches on cache performance,” in *International Conference on Architectural Support for Programming Languages and Operating systems (ASPLOS)*, 1991.

11. G. E. Suh, E. Peserico, S. Devadas, and L. Rudolph, "Job-speculative prefetching: Eliminating page faults from context switches in time-sharing systems," 2001.

12. D. Chiou *et al.*, "Scheduler-based prefetching for multilevel memories," 2001.

13. P. Koka and M. H. Lipasti, "Opportunities for cache friendly process scheduling," 2005.

14. C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Workshop on Experimental Computer Science*, 2007.

15. D. Tsafrir, "The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops)," in *Workshop on Experimental Computer Science*, 2007.

16. F. M. David, J. C. Carlyle, and R. H. Campbell, "Context switch overheads for Linux on ARM platforms," in *Workshop on Experimental Computer Science*, 2007.

17. A. Agarwal, J. Hennessy, and M. Horowitz, "An analytical cache model," *ACM Trans. Comput. Syst.*, vol. 7, no. 2, pp. 184–215, 1989.

18. G. E. Suh, S. Devadas, and L. Rudolph, "Analytical cache models with applications to cache partitioning," in *International Conference on Supercomputing (ICS)*, 2001, pp. 1–12.

19. G. E. Suh, S. Devadas, and L. Rudolph, "A new memory monitoring scheme for memory-aware scheduling and partitioning," in *International Conference on High Performance Computer Architecture (HPCA)*, 2002, pp. 117–128.

20. W.-m. Hwu and T. M. Conte, "The susceptibility of programs to context switching," *IEEE Transactions on Computers*, vol. 43, no. 9, pp. 994–1003, 1994.

21. F. Liu and Y. Solihin, "Understanding the behavior and implications of context switch misses," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 4, pp. 21:1–21:28, 2010.

22. H. Cui and S. Sair, "Extending data prefetching to cope with context switch misses," in *International Conference on Computer Design (ICCD)*, 2009, pp. 260–267.

23. J. A. Brown, L. Porter, and D. M. Tullsen, "Fast thread migration via cache working set prediction," in *International Conference on High Performance Computer Architecture (HPCA)*, 2011, pp. 193–204.

24. M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for MLP-aware cache replacement," in *International Symposium on Computer Architecture (ISCA)*, 2006, pp. 167–178.

25. M. K. Qureshi, "Adaptive spill-receive for robust high-performance caching in CMPs," in *International Conference on High Performance Computer Architecture (HPCA)*, 2009, pp. 45–54.

Introduction

Welcome to the sixth edition of the VMware Technical Journal. As we have covered in previous editions, VMware has three primary focus areas: the software-defined data center (SDDC), hybrid cloud, and end-user computing (EUC). Thus far our focus in these journals has been primarily on the SDDC and its vast and wide array of technologies. For this issue we wanted to expand our focus and spend time discussing all the innovation that's happening in the EUC space. Although VMware has been in the EUC space for the past seven or eight years, it's only in the past few years that we've taken a more expansive view—including not only virtual desktop, but also physical desktop, enterprise mobility management, identity, social, application delivery, and much more.

The EUC team's mission is to enable a secure virtual workspace for work at the speed of life. The reality is that consumerization of IT is bringing more—and more diverse—devices onto company networks. The "one size fits all" one-desktop-per-employee model no longer works. IT now needs to manage a plethora of different devices, enabling rapid delivery of a user's applications and data to all those devices while at the same time ensuring security and compliance. Users, on the other hand, are demanding a seamless, integrated experience. They want information and apps at their fingertips and want to be able to set down one device, pick up another, and start right where they left off. These are some challenging requirements!

With that in mind, we've been driving much innovation in the EUC space at VMware. As I mentioned above, security is a critical area here, and we have papers that address two different aspects of security: lightweight mobile device authentication using QR codes, and enhanced data center network security through integration of AirWatch® by VMware and VMware NSX™ for context-based policy enforcement. On the desktop front, we're reimagining virtual desktop with our concept of just-in-time desktops, enabling radical reduction of costs and simplification of management. We're also getting very creative with enterprise social. We present work on how we can help users find relevant information faster by cross-referencing their social graph and their file graph. We also experiment to see if social technologies can help enhance performance management (spoiler alert: they can!). Finally, analytics plays a key role across our products, and we present our findings for optimized data warehousing using NoETL technologies.

As you can see, there's a tremendous amount of exciting innovation happening within the EUC space at VMware. We hope you enjoy this issue of the VMTJ and, as always, we welcome your comments and feedback.

Kit Colbert
VMware Principal Engineer
End-User Computing

The Role of Social Graph in Content Discovery Within Enterprise Social Networking

Niloufar Sarraf
Socialcast by VMware
nsarraf@vmware.com

Abstract

One of the challenges for employees in the workplace is to be able to seamlessly discover relevant useful content within the massive quantity of files that are constantly generated and shared [4]. Industry research suggests that social search makes it more likely that end users will find the information that they need [5, 7]. In this study, we aimed at examining Socialcast® end-user habits, workflows, pain points, and needs in regard to content sharing, discovery, and collaboration. For the purpose of this study, we conducted structured interviews with 20 participants as well as prototype testing. Our study provided deep insights into factors that influence enterprise social networking content discovery, the traditional applications used, content metadata, and the most useful content search categories.

Our findings suggest that social graph, along with company culture, does affect end-user habits, workflow, and behavior when it comes to discovering and collaborating on content in the workplace.

Keywords: social graph, content discovery, enterprise social networking

1. Introduction

Socialcast, as an enterprise social networking (ESN) platform, aims to help employees get their jobs done by providing efficiency and productivity tools, a superior user experience, and a ubiquitous architecture. Socialcast helps companies work more effectively through the seamless capability to find and discover relevant useful information in the workplace. For Socialcast to accomplish these goals, the research team constantly works toward gaining better understanding of end-user workflows, pain points, habits, and behavior.

Kit Colbert, in his talk "Bringing VMware and AirWatch Together," suggested several points of connection between content discovery and social graphs [1]. Colbert proposed that social graphs be embedded in all applications, services, and devices. More particularly, Colbert suggested embedding social information within files and data to identify social graphs that use particular content. He said, "You can get a better idea of who to collaborate with on a certain topic from that information." Moreover, in regard to social content sharing and collaboration, he suggested cross-referencing the social network and the file/data network, which would give the end users the opportunity to know who had been accessing their files as well as the social connections who had collaborated on any given content. This, he meant, would make it easier for end users to know whom to work with on a given issue, because the information would be readily available on the Socialcast home stream.

Our aim in this study was to investigate the relationship between social graph and content discovery and to investigate the way in which these affect content sharing and content discovery [8]. The next section lists the main research objectives of this project.

2. Research Objectives

The study's research objectives are to

- Gain understanding of end-user habits, workflows, pain points, and needs in regard to content sharing and discovery solutions
- Gain understanding of end-user conversations and collaborations relating to content
- Determine the most relevant/useful content metadata in regard to content discovery

3. Study Phases

Before conducting our study, we looked at the current literature on social graphs and content discovery within ESN platforms. Based on the results of this initial research, and to gain a holistic understanding of our objectives, we divided the study into two phases: preliminary and fundamental.

3.1 Preliminary Study

The main purpose of the preliminary part of the study was to gain insights into end-user content-sharing habits, workflows, pain points, and needs. Moreover, we tested end-user content-discovery behavior and interaction with prototypes, aiming to gain insights into an initial design solution.

3.2 Fundamental Study

Based on the key findings of the preliminary research projects, we conducted a second phase of the study with the intention of gaining deeper understanding of end-user daily work flows, constant pain points, and burning needs in regard to content sharing and users' social graphs on Socialcast.

Next, we provide an overview of the literature review and each of the two study phases.

4. Literature Review

IBM [7] studied discoverability of files in social content sharing solutions. One of the main research questions the study sought to answer was: "How do users decide which files to download?" To find the answer, they tested content-sharing solutions that displayed a "file page." The file page was meant to provide a summary of content shared, along with activities that were related to the files. In this study, the researchers focused on end users who had no prior knowledge of the existence of content shared on the ESN.

Figure 1 (originally published in [7]) shows the six navigation pathways that users can follow to discover a file, along with how much they were used during the study period. After discovering a file, users evaluate the file and decide whether or not to download it. The findings include but are not limited to:

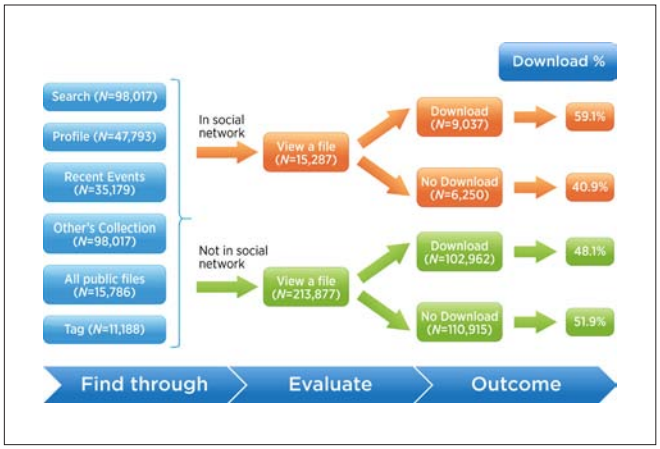


Figure 1. File-Browsing Model. N = Total Number of Events in That Category. (Source: Shami et al., Just a Click Away: Social Search and Metadata in Predicting File Discovery, Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media, 2011.)

- Social file sharing complemented by metadata can enhance information search and discoverability.
- The likelihood of file download increases when
  - The file's author is in the user's social network.
  - The file is downloaded or shared by the user's networks.
  - The file's metadata matches the user's interest.
- Social information has a strong influence on user behavior.
- File discoverability increases when
  - The file is given a proper title, description, and tag.
  - File content is combined with the file's social graph information.
  - The file-sharing solution actively reaches out to other software that is connected or related to the file-sharing solution.
- Social networking increases the likelihood of discovering content.
- Utilizing the social graph has the potential to improve the ability to uncover content and people/groups.

Other industry studies corroborate and support the findings in [7]. These studies also suggest that social graphs provide tremendous value when it comes to both discovering content and collaborating on content [2]. Moreover, we found studies indicating that social sharing of relevant useful content, while making content more likely to be discovered, contributes to increased user engagement [3]. This is an important factor, especially for lurkers who represent the majority of end users in any ESN platform [5].

5. Methods

For the preliminary study, we conducted a mixed-methods study that consisted of structured interviews and a Rapid Iterative Testing and Evaluation (RITE) study. The structured interview aimed to establish a primary baseline for end-user workflows, paint points, and habits. The RITE study was designed to test the low-fidelity prototype. The results of this study, although extremely insightful, helped us realize that additional in-depth data was needed to gain deeper insights into Socialcast end users.

As a result, as a follow-up study, we designed a more elaborate structured interview that helped us gain deeper understanding of Socialcast end users and their daily workflows and consistent pain points pertaining to content discovery and content sharing in the workplace.

Next, we provide an overview of the key findings.

6. Key Findings

6.1 Content Solutions Used

When it came to the type of content solutions being used, we found that the participants tend to use the same types of file solutions that their team, stakeholders, or company uses. End users exhibited tendencies toward complying with the company culture.

Among these content solutions, the participants reported having different preferences that were contingent upon the nature of the way in which the content was being used. For example, when it came to content that was in its final stages of development, the participants chose repositories that offered robust archival types of solutions. On the other hand, for in-progress types of content, the participants used solutions that were more likely to allow for collaboration. Furthermore, the majority of the companies were strict in using solely company-approved solutions, while others were not as rigorous in implementing company policies.

6.2 Content-Discovery Habits

In regard to content-discovery habits, the majority of the participants exhibited traditional behavior, which consisted of primarily using email applications (and sometimes meetings). In other words, the participants were more likely to share and discover content within email applications. The participants not only depended on email applications but also relied heavily on mental notes. That is, they tried to remember events, conversations, and solutions based on their own memory or those of their coworkers.

Interestingly, the company culture and the environment appeared to have a strong influence on end-user behavior when the enterprise social graph appeared to impact user behavior either directly or indirectly. To illustrate, it was more likely for the internal and the external teams to set trends on the basis of the types of content-repository solutions that were mostly used. However, it is worth noting that these trends tended to be fluid and could change over time, if enough people in one's social graph followed the same trend.

6.3 Conversations About Content

We were also interested in learning how and where end users carry on conversations when sharing and retrieving content. One of our findings indicated that, here again, the end users mainly use email applications to exchange, retrieve, and hold conversations about content. In fact, email applications appeared to be the main repository for content, conversations, and activities.

Additional places where these conversations occurred were at (in-person or remote) meetings, in instant messaging, and sometimes on the phone. The differentiating characteristics of the usage of each solution type appeared to be contingent upon the context. For example, meetings were scheduled for projects that needed immediate actions or were complex; the complexity of the project was determined by the number/type of employees, time zones, project type, deadlines, and so on.

Few participants reported using instant messaging with their social graph for quicker content search and conversations. On the other hand, a few indicated that it was calling close teammates that helped them find, collaborate on, and converse about content the quickest.

6.4 Content Discovery Workflow

Although it was the type of project, teams, number of collaborators, time zones, and so on that determined the choice of the software solutions, the daily workflows appeared to start and end within email applications regardless of the situation. The following is a step-by-step daily workflow (also see Figure 2):

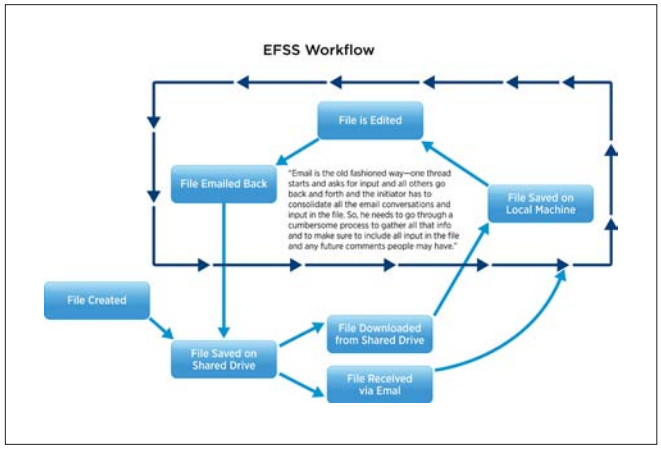


Figure 2. Content-Discovery Workflow

1. A file is either
   - Created and saved on a shared repository.
   - Downloaded from a shared repository to the local machine.
   - Received via email or a file-transfer solution and downloaded to the local machine.
2. The file is edited and updated.
3. The file is emailed to colleague(s) for feedback or approvals.
4. The recipient saves the file to the local machine (or on the shared drive) to review and to provide feedback on the file.
5. Feedback is provided as
   - Text in the body of emails.
   - Tracked changes within the document.
   - Comments within the document.
6. Conversations about the file occur as text (sometimes in different colors/fonts) within the body of emails.
7. The file is emailed back to the sender and/or group(s) of colleagues for additional feedback and/or approval.
8. The final version of the file is uploaded to file repositories.
9. File versions are tracked manually:
   - Manual (Excel) sheet created for tracking purposes only.
   - Search within email applications.
   - Mental notes.

As one can imagine, this workflow has a tendency to become quite cumbersome and time-consuming in the workplace. As a result, a file's life cycle can suffer and became unnecessarily complex and inefficient. To illustrate, content life cycles can end up having these four stages:

1. Starts locally as either a team template or an individual file.
2. Shared on repositories for teams to access.
3. Downloaded from the shared repositories to the local machine.
4. Edited and uploaded back to the shared repositories.

We believe that one of the participants expressed this cumbersome workflow perfectly:

"Email is the old fashioned way—one thread starts and asks for input and all others go back and forth and the initiator has to consolidate all the email conversations and input in the file. So, he needs to go through a cumbersome process to gather all that info and to make sure to include all input in the file and any future comments people may have."

6.5 Motivation and Decision Making

For working with content, specific factors seemed to impact the end-user decision-making processes more than others. These factors were directly (or indirectly) related to the enterprise social graph. Participants discovered content based on

- **Team members and stakeholders** – The repository that was already used by teams or stakeholders.
- **User-friendliness** – Ease of accessing, obtaining, and maintaining the repository space.
- **Trust** – The reliability of the repository space when it came to retrieving content.
- **Metadata**

Metadata turned out to be one of the crucial points when it came to content-related decision-making factors. It is important to note that, at the time when we conducted this study, the participants tracked the file metadata mainly through mental notes or manual

tracking sheets, such as an Excel sheet. Although the participants found the manual tracking of metadata to be extremely inefficient, they still ended up collecting this data manually simply because they could not find any other solution. This manual process, in turn, made it hard to discover relevant content in a timely manner.

One of the main benefits of metadata was for the participants to be able to determine "who did what when." Discovering the answers to these questions through metadata not only had the potential to increase the likelihood of discovering content, but it also helped end users work more efficiently.

The participants reported these metadata to be the most useful:

- File description
  - Would this information be useful to me?
- Original author
  - Do I know this author? Is he/she a coworker?
  - How am I connected to this person?
- Date created
  - How accurate is this content? Would it be useful to me?
- File size
  - How long will it take for me to download this content?
- File type
  - Which software would I need to open this file? Do I have the software or do I need to first get it before downloading this file?
- Number of file downloads/views
  - Is this a popular type of content?
  - Are there any people that I know who downloaded it?

6.6 Content Search

6.6.1. Search Based on People (Social Graph)

It was more likely for the participants to recall people who were associated with a piece of content. Therefore, in a search for content, the initial search keywords tended to be people names rather than content or project names. When trying to discover and search for content or the conversations about it, the participants were more likely to start their searches with the names of employees that they, for example, had worked with on a particular piece of content.

6.6.2. Search Based on Content Topic

If the participants did not recall the names of the employees related to specific content or if the keywords related to the employees failed, as a next step they tended to search based on the topic or the name of the content or the project.

7. Conclusion

Our research not only corroborated the industry findings but also demonstrated the tremendous value of social graphs of Socialcast in content discovery and collaboration. This is an important factor, especially for lurkers who represent the majority of end users in any ESN platform [5].

More specifically, our findings suggest that social graphs, along with company culture, affect end users' daily workflows and behavior in the workplace. Lastly, we established that metadata is an important part of files and has great potential in promoting content discovery. The most useful metadata, as also indicated in the industry literature [7], are description, author, date, size, type, and number of views/downloads.

These findings strongly suggest design implications for promoting the Reader-to-Leader framework [6]. One design implication is to build recommender systems that are based on users' existing social graph activities in regard to content sharing. Based on these activities, the recommender system will recommend the most relevant, useful, and popular content to the end users. Moreover, the recommender system could be used to potentially increase user engagement by encouraging users to connect with one another, based on their content-sharing types and behavior. This, in turn, has the potential to contribute to increased user engagement in terms of conversations about files and content. Hence, employees will not only have increased chances of discovering relevant and useful content, but they will also have the opportunity to connect with other employees in their fields of interest, which can contribute to increased efficiency and collaboration in the workplace.

Acknowledgments

Big thanks to Jens Koerner, Oren Root, and Badi Azad for their support and help with our user experience research projects and efforts at Socialcast.

References

1 Kit Colbert (2014). Bringing VMware and AirWatch Together. https://www.youtube.com/watch?v=p8uzevtEPHs

2 Farnham, S. D., Turski, A., and Halai, S. (2012). Docs.com: Social File Sharing in Facebook. FUSE Labs, Microsoft Research.

3 Forrester Research (2014). Social Technology and the Groundswell of Consumer Adoption. http://www.swmediagroup.com/social-technology-and-the-groundswell-of-consumer-adoption/

4 Jensen, C., Lonsdale, H., Wynn, E., Cao, J., Slater, M., and Dietterich, T. G. (2010). The life and times of files and information: a study of desktop provenance. *Proc. CHI*. ACM Press.

5 Muller, M., Shami, N. S., Millen, D. R., and Feinberg, J. (2010). We are all Lurkers: Consuming Behaviors among Authors and Readers in an Enterprise File-Sharing Service. http://www.watson.ibm.com/cambridge/Technical_Reports/2010/TR2010.11%20We%20are%20all%20lurkers.pdf

6 Preece, J. and Shneiderman, B. (2009). The Reader-to-Leader Framework: Motivating Technology-Mediated Social Participation, *AIS Transactions on Human-Computer Interaction*, (1) 1, pp. 13–32.

7 Shami, N. S., Muller, M., and Millen, D. (2011). Just a Click Away: Social Search Metadata in Predicting File Discovery. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/viewFile/2854/3271

8 Thom-Santelli, J. and Millen, D. (2010). Characterizing Social Data Sets: Why So Hard to Share? http://www.academia.edu/2724751/Characterizing_social_data_sets_Why_so_hard_to_share

NoETL: ETL Code Generation for a Dimensional-Data Warehouse

Michael Andrews
Socialcast by VMware
andrewsm@vmware.com

Abstract

In modern data-warehousing environments, Extract-Transform-Load (ETL) tools and practices still remain an unnecessary bottleneck for agile and iterative business intelligence [1]. Many tools simply address the transformation phase of the ETL process and fail to take advantage of the structure and representation of the end-goal reporting database. In this paper, we discuss at length the deficiencies of current ETL systems as well as best practices for dimensional modeling. Finally, we describe an approach to ETL in which a simple high-level specification generates the end-goal reporting database in addition to the code required for executing an ETL data pipeline.

Keywords: ETL, data warehouse, dimensional modeling, Hive

1. Introduction

Most modern information systems generate large amounts of data—data that is often useful to business analysts and data scientists for supporting business decisions and formulating new hypotheses.

Given the relatively cheap cost of storage, as well as the ease of initial implementation, many systems are designed to simply store all possible sources of data within a central, read-only repository. Typically, this initial raw-data repository is not amenable to analysis and must be organized to support data-driven business intelligence. A common approach to this problem involves leveraging tools for extracting, transforming, and loading raw source data into a database that is optimized for running reporting queries [4]. Although existing ETL tools and methodologies provide a time-tested approach for constructing a reporting database, many do not take advantage of the underlying structure of the reporting database to simplify development [6].

2. Traditional ETL

A traditional ETL-based system is constructed as a long pipeline of scripts that extract, transform, and ultimately load the data into a reporting database. The first challenge for any ETL data pipeline is to cleanse and convert various data sources into a homogeneous format to prepare for transformation. Many common types of fields in the raw data must be encoded in a uniform fashion before data transformation can occur (e.g., timestamps denoting data set creation and modification). Additionally, data formats vary over time, resulting in complex logic for cleansing and converting data, which can drastically increase maintenance costs. Typically, data that has previously been converted and transformed must later be reprocessed due to changing requirements or programming error [5].

Historical data sets are typically very large and therefore must be processed in large batches and in parallel for performance and scalability. For these reasons, in modern-day ETL systems, programmers typically try to bring the computation to the data by first loading raw data into a distributed storage and processing environment (e.g., Hadoop) or a massively parallel relational database (e.g., Greenplum) before the transformation stage. The types of transformations available at this stage are limited by the datastore itself and are typically extremely verbose—for example, Java MapReduce code or large SQL queries, which further compound the original complexity of a large number of changing data formats.

Outside of the performance constraints of dealing with large amounts of data, analysts expect that new data is processed in a timely fashion, and that batch updates to historical data sets do not impact their day-to-day activities. These issues, inherent in the problem domain, typically lead to the development of separate pipelines for near real-time processing and batch processing [2]. Together, the large number of data formats, the necessity for verbose transformation code, and the need for both near real-time and batch-processing pipelines create unnecessarily long development cycles that prevent the business analyst from iterating on the problem at hand.

3. Reporting Database

Before we discuss our approach to ETL—the end goal of the process, the reporting database and its structure must first be considered. The primary function of a reporting database is to enable the business analyst to join domain-specific knowledge with operational insights in an ad-hoc fashion. To facilitate this type of analysis, two key types of information tables are required: *dimension tables* containing domain-specific attributes and *fact tables* containing numeric operational measurements. These types of tables are naturally organized in a star or snowflake schema with long and narrow fact tables at the center referenced by short and wide dimension tables on the periphery. In addition to the traditional star schema layout, many data warehouse designers

adhere to the tenets of dimensional modeling outlined in detail by Ralph Kimball [3]. The dimensional model is a series of best practices, conventions, design patterns, and a domain-specific language to describe many of the common problems, pitfalls, and approaches associated with constructing a data warehouse.

3.1 Dimension Tables

In the dimensional model, *dimension tables* are categorized into a variety of types describing object-attribute change data capture (CDC). In a *type 1 dimension table*, new object attributes overwrite previous attributes, similar to how changes are recorded in an operational context. For the purposes of a reporting database, these types of tables are useful for modeling "mini dimensions" that are used to represent unbounded enumerated attributes where a historical record is not relevant—for example, enumerating UserAgent strings. A *type 2 dimension table*, or slowly changing dimension (SCD), records periodic historical snapshots of an object with additional metadata representing the effective data range of the record as well as the totally ordered version of the record. Because the operational primary key for the object is not expected to vary over time, an additional surrogate key is used to label the object at particular point in time. This representation is particularly useful for historical reporting because it enables the analyst to select attributes at a point in time and join on associated operational measures in a fact table (Figure 1). Finally, a *type 4 dimension table* records the entire history of an object, including data-provenance information and whether or not the modification record is a complete snapshot or a partial delta record. This allows the system to ingest multiple sources of partial data, which is then consolidated into a single periodic snapshot record of the object stored in a type 2 dimension table.

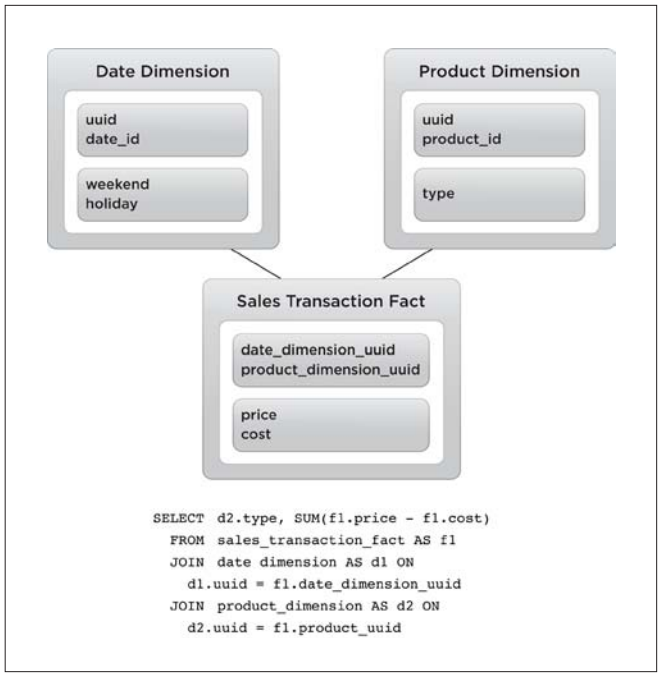


Figure 1. Total Profit by Product Type on Holidays

3.2 Fact Tables

Fact tables record operational measurements at a particular granularity as well as references to dimension tables. *Transaction fact* tables record operational measurements at the lowest level of granularity—typically at the millisecond level. For efficient reports, transaction fact tables are rolled up to higher levels of granularity using an aggregate function defined by the measure to form a *periodic snapshot fact table*—for example, "the total number of unique visitors to a web page by hour, day, and month." For some measures it is often useful to accumulate the aggregate function in an *accumulating snapshot fact table*—for example, the "cumulative number of active users over time." This enables a business analyst to quickly perform ad-hoc queries over cumulative measures for a particular date range without first performing an expensive summation starting from the beginning of time.

4. Goal-Based ETL

Our approach to ETL begins with the important observation that the structure of the reporting database not only describes how to efficiently formulate reports but also describes a process for iteratively loading and updating new and existing data. Following this observation, we encode the goal-reporting database, raw-data sources, and transformations in a domain-specific language (DSL) based on the dimensional model (Figure 2). This DSL is leveraged both to define the data schema (DDL) and generate data manipulation language kernels (DML) to process the data in batch and near real-time. In practice, our compact DSL, when compared to generated DDL and DML kernels, requires fewer than 20x lines of code (LOC) for dimension operations and fewer than 30x LOC for fact operations.

```
dimension 'user', type: :four do
 references :department, insert: true
 references :user_account_state
 column 'user_account_state', type: :enum,
   values: %w(registered inactive active),
   default: 'registered'
 column 'tenant_id', type: :integer,
   index: true,
   natural_key: true
 column 'user_id', type: :integer,
   index: true,
   natural_key: true
end
```

Figure 2. High Level NoETL Dimensional DSL

From the high-level specification, actual DDL statements for various datastores are generated, along with metadata tables and columns used for bookkeeping by the DML kernels (Figure 3). Source data sets are transformed by high-level functions that specify a map between

input rows and output rows. Depending on the datastore and the size of the input data set, map functions are either executed within the data-processing environment (e.g., Hadoop streaming) or outside the environment before data is loaded. Source and target data sets are partitioned by creation time and labeled with a last-known-modification timestamp. To prevent unnecessary reprocessing, a dependency tree of source data sets and target data sets is inferred from the DSL and leveraged to reprocess only source data sets whose modification times are greater than the target data set modification times. This data flow processing approach, similar to GNU Make, has the advantage of naturally unifying the near real-time and batch-processing pipelines by allowing the batch granularity to vary depending on the context, without modifying the update rules. For example, a near real-time pipeline can be constructed to operate at the hourly level of granularity, over the last month of data every half-hour, whereas the same update rules can be reused to construct a batch-processing pipeline that operates at a monthly level of granularity over the entire historical data set once per day (Figure 4) Additionally, the partitioned data sets, in conjunction with the data-dependency tree, allow for the intuitive construction of a naively parallelizable data-processing plan composed of sequential DML kernels for transforming, loading, and updating target data sets.

```
class VisitationFact < Thor
  namespace :visitation_fact

  desc 'daily_transaction',
   'Compute daily visitation transaction
fact table from hourly request logs'
  target path: fs.path(:warehouse_dir,
'daily_visitation_fact/y=%Y/m=%-m/d=%-d')
  source path: fs.path(:repository_dir,
'requests/y=%Y/m=%-m/d=%-d/h=%-k')
  def daily_transaction_task
    targets.missing.actionable do |target|
     hive file: fs.path(:flow_dir,
'transaction_fact_job.hql'), \
        variables: {
          Y: target.start_date.year,
          M: target.start_date.month,
          D: target.start_date.day }
    end
  end
  ...
end
```

Figure 3. High Level NoETL Data Dependency DSL

4.1 Load Operation

The kernels generated for data loading are simple and adhere to the best practices for the underlying data store.

In an RDBMS, such as Postgres, source data is first "staged" into an empty copy of the target table and then loaded in batch using the common bulk upsert operation (update or insert). For datastores

```
CREATE TYPE USER_ACCOUNT_STATE_TYPE AS ENUM
 ('registered', 'inactive', 'active');

CREATE TABLE IF NOT EXISTS user_dimension_ledger
(
 ...
);

CREATE TABLE IF NOT EXISTS user_dimension
(
 uuid UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
 department_type_uuid UUID NOT NULL
   REFERENCES department_type(uuid)
   DEFAULT default_department_type_uuid(),
 user_account_state USER_ACCOUNT_STATE_TYPE NOT NULL
   DEFAULT 'registered',
 tenant_id INTEGER NOT NULL,
 user_id INTEGER NOT NULL,
 parent_uuid UUID
   REFERENCES user_dimension_ledger(uuid),
 record_uuid UUID
   REFERENCES user_dimension_ledger(uuid),
 start_at TIMESTAMP NOT NULL DEFAULT TO_TIMESTAMP(0),
 end_at TIMESTAMP,
 version INTEGER DEFAULT 1,
 last_modified_at TIMESTAMP NOT NULL DEFAULT NOW()
);
```

Figure 4. Generated SQL Dimensional DDL

that do not have row-level ACID properties, such as Hive, or when such operations are not efficient (e.g., large partitioned tables), the entire source data set is staged in a "shadow" partition and subsequently loaded into the target data set via an atomic partition rename. This allows long-running batch operations to update the reporting database without significantly impacting the day-to-day activities of the business analyst or cause downtime for reporting dashboards.

4.2 Type 4 Dimension Update Operation

There are several steps in updating a type 4 dimension table. First, new or modified raw-data sources must be loaded into the type 4 dimension table. A data source is considered stale and marked for reprocessing if the modification time exceeds the modification time of the target data set, in this case the `MAX{dimension_table.last_modified_at}`. After a data source is transformed and staged, it is loaded into the type 4 dimension table via a bulk upsert operation, which updates the `last_modified_at` as a result. It should be noted that each new data source or data-source partition can be loaded into the target type 4 dimension in parallel without fundamentally changing the update-operation semantics. Next, the type 4 dimension table is "consolidated" into an empty stage table copy of the target type 2 dimension table. The consolidation process generates a series of periodic snapshot records by merging complete records with their respective partial-update records via a window function.

In addition to effectively merging scalar columns, the consolidation process crucially is able to correctly merge complex data types such as key-value columns (e.g., `hstore`) which is achieved using a user-defined aggregate function. The resulting type 2 dimension stage table is then bulk upserted into the target type 2 dimension table. After the target type 2 dimension table is consolidated, the

`start_at` and `end_at` columns that form the effective date range are "relabeled" to form nonoverlapping intervals, and the totally ordered `version` column is updated accordingly (Figure 5). A key property of the dimension-update operation is that it is idempotent, and it preserves the integrity of the surrogate key relationship. Therefore, if a fact table references a dimension table via a surrogate key, this relationship itself is not altered, though late-arriving updates can be applied to the effective dimension record in a consistent manner.

TENANT_DIMENSION_LEDGER				
tenant id	attributes	source	created at	delta
1	{A:1, B:2}	snapshot	T_0	0
1	{B:2}	event	T_1	-1
1	{A:2}	event	T_1	+1
⋮	⋮	⋮	⋮	⋮
2	{A:1}	snapshot	T_1	0

TENANT_DIMENSION				
tenant id	attributes	start at	created at	version
1	{A:1,B:2}	T_0	T_1	1
1	{A:2}	T_1	∅	2
⋮	⋮	⋮	⋮	⋮
2	{A:1}	T_1	∅	1

Table 1: Merge, Consolidate, Relabel

4.3 Transaction Fact Update Operation

Transactional fact tables need to be updated whenever downstream data sources arrive or have been modified. Transforms are typically fairly simple and consist of extracting measures and natural keys from raw-data sources. Occasionally, more-complex logic is required—for example computing interarrival times or sessionizing data based on domain-specific rules. For these types of computations, our DSL allows custom MapReduce kernels to be applied to the source data set before staging the resulting target intermediate fact table. Once the intermediate fact table has been staged, the natural keys must be resolved to dimensional table keys using a simple join operation that takes into account the effective date of the dimension record. Completed "shadow" fact tables are then loaded into the data warehouse via an atomic partition rename operation. Transaction fact tables must also occasionally be "relabeled" when referenced dimension records in the partition have been inserted due to a type 4 dimension update operation. *Periodic snapshot fact tables* and *accumulating snapshot fact tables* are computed from downstream transaction fact tables using the same modification-time comparison rule. In the case of *accumulating snapshot fact tables,* some effort is required to reduce unnecessary churn, because all upstream target partitions after the first "stale" downstream source partition need to be recomputed for correctness.

5. Future Work

In theory, the ETL process described can be applied to any relational datastore, though in practice we have chosen to generate Postgres and Hive DDL and DML kernels. Postgres kernels were chosen for dimension-update operations due to the ease of implementation when ACID properties are leveraged with additional analytic windowing functions. Hive kernels were chosen for fact load and transformation operations mostly due to the large size of fact tables and the easy parallelism gained by using Hadoop. Some amount of effort has been required to bridge the gap between the two chosen datastores, and we have not fully explored the space of strategies available to us in this regard. For example, although fact tables are computed by Hive kernels, natural key resolution occurs in Postgres by first loading Hive partition tables into Postgres partition tables and then performing natural key resolution. The natural key resolution operation could easily be done at scale via a Hive kernel that joins a snapshot of the dimension tables to the intermediate fact table. Additionally, data shuffling between Hive and Postgres could be alleviated by exposing Hive tables over JDBC to a Postgres Foreign Data Wrapper (FDW) external table. Ideally, it should be possible to generate kernels for a single datastore that has the necessary ACID properties required for dimension-update operations, as well the easy horizontal scalability of Hive for large fact table computations.

6. Conclusion

Large data warehouses require ETL processes that transform raw source data into format suitable for reporting queries. We have discussed various deficiencies with traditional ETL-based tools, in contrast to our approach to ETL. Additionally, we have demonstrated that leveraging the goal-reporting database structure enables code generation and automation, which reduce maintenance cost and the time to iterate on the problem domain.

Acknowledgments

Thanks to Jeff Sato and Justin Portillio for their valuable feedback as data scientists and business analysts, to Lilit Div for managing and supporting the project, and to Jens Koerner for leading the Socialcast analytics initiative at Socialcast.

References

1. Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. Mad skills: new analysis practices for big data. Proceedings of the VLDB Endowment, 2(2):1481–1492, 2009.
2. Thomas Jorg and Stefan Dessloch. Near real- time data warehousing using state-of-the-art ETL tools. In Enabling Real-Time Business Intelligence, pp. 100–117. Springer, 2010.
3. Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.
4. Celina M. Olszak and Ewa Ziemba. Approach to building and implementing business intelligence systems. Interdisciplinary Journal of Information, Knowledge, and Management, 2:134–148, 2007.
5. Alkis Simitsis, Dimitrios Skoutas, and Malu Castellanos. Natural language reporting for ETL processes. In Proceedings of the ACM 11th International Workshop on Data Warehousing and OLAP, pp. 65–72. ACM, 2008.
6. Panos Vassiliadis. A survey of extract–transform–load technology. International Journal of Data Warehousing and Mining (IJDWM), 5(3):1–27, 2009.

A Framework for Secure Offline Authentication and Key Exchange Between Mobile Devices

Erich Stuntebeck
AirWatch
estuntebeck@air-watch.com

Kar-Fai Tse
AirWatch
ktse@air-watch.com

Chaoting Xuan
AirWatch
cxuan@air-watch.com

Chen Lu
AirWatch
chenlu385@air-watch.com

Abstract

As mobile devices grow ever more common and embedded in our daily lives, carrying multiple devices, such as a smartphone and a tablet, is also becoming more common. This work proposes a solution that leverages this phenomenon to provide a high level of protection for data-at-rest on a mobile device. A time-varying QR code containing an encryption key protected by a preshared secret known to both devices is displayed on a designated authenticator device. The device on which the user is to be authenticated, and decrypted content made available, uses its camera to capture that code and extract the key needed to decrypt its stored content. The decryption key is never stored to disk and is erased from memory when no longer needed or after a specified timeout. This system makes the encrypted data stored on the one device useless unless the device containing the key for its content is also present and unlocked.

1. Introduction

Mobile devices have become part of our everyday lives. Smartphones are nearly ubiquitous, and tablets, with their larger-format displays suitable for longer duration and more-involved tasks, are not far behind. It is becoming more and more common to carry both a smartphone and a tablet, and with them ever-increasing amounts of potentially sensitive data.

Whereas protecting sensitive data on laptops and desktops is reasonably straightforward with the use of Full Disk Encryption (FDE), the same is not true on mobile devices. FDE typically works by protecting a master encryption key for the disk with a password-derived key. Although FDE can utilize strong encryption, such as AES with 256-bit keys, the encryption is only as strong as the key. Because the key is often derived from a password, the password is FDE's weak link. If a password is easily guessable or of insufficient length and complexity to make a brute-force attack infeasible, the encryption key and data can potentially be recovered by an attacker.

On desktops and laptops, password complexity requirements are typically enforced by enterprise IT departments, leading to passwords that should be of sufficient complexity to support a password-derived key for FDE. Although the same requirements can be enforced on mobile devices via the use of Mobile Device Management (MDM) software, they often are not, due to the inconvenience of entering a complex password on a mobile device. Additionally, mobile devices are often used for short-duration interactions, and as such require authenticating many more times throughout the day than a user would typically need to do on a desktop or laptop. Given this, the prevalence of simple four-digit passcodes, or even no passcode at all, on mobile devices should come as no surprise. A four-digit passcode offers only $\binom{10}{1}^4=10{,}000$ possible combinations. Although password-based key-derivation algorithms such as PBKDF2 can lengthen the brute forcing process by adding computational complexity through multiple rounds of hashing, this can delay the inevitable by only so much with so few combinations. Security researchers have shown that even with added security measures such as key entanglement with device-specific UID keys, an encryption key based on a four-digit passcode can be brute forced in as little as 18 minutes [1] [2].

Although one option for protecting data on a mobile device is to treat the device as a dumb terminal and never to store to disk anything considered sensitive, the realities of spotty network connections, data usage caps, and roaming charges when travelling internationally make pulling all data from a remote server as needed an unappealing option. Given this, we set out to develop a system to protect data-at-rest stored on a mobile device with strong encryption and a key that is independent of any user password or device hardware keys.

2. Related Work

Authentication systems are often described in terms of having one or more of three components: something you know (passwords), something you have (tokens), and something you are (biometrics). We have already addressed the weakness of passwords on mobile devices, and in this section we discuss related work for the other two.

2.1 Mobile Multifactor Authentication

Hardware token–based multifactor authentication was popularized by the RSA SecurID tokens, which provide a continually changing numeric passcode typically required in conjunction with a password to authenticate to a server [3]. SecurID and other similar tokens typically utilize some variant of the HMAC-based One Time Passcode (HOTP) or Time-Based One-Time Passcode (TOTP) algorithms [4] [5] . These algorithms typically produce a short numeric code based on truncating a hash of a shared secret known to both the token and a server and either a counter (HOTP)

or the current time (TOTP). Both the token and the server calculate the expected current value of the passcode; to authenticate, the server validates the user's input against the expected value.

Systems like this were designed for authentication to network services, not local devices. The critical distinction is that network services run on physically secured hardware. In a properly implemented system, if authentication fails, there should be no possibility of accessing secured data. In this situation, challenge/response or one-time passcode validation can serve as a useful second factor to passwords. Mobile devices need stronger protection for data-at-rest. These devices are by their nature **mobile**, and as such data must be protected even in the event of physical compromise. In this situation, authentication via validation of a code from a second factor is insufficient. With physical access to a device, any software guards requiring an HOTP or TOTP code could potentially be bypassed because the codes are not cryptographically related to the key used to protect stored data on the device. The cryptographic keys used to encrypt data on a device must be obtained from or at least derived from data not stored on that device.

Smart cards are another potential second authentication factor for mobile devices. Smart cards contain a simple processor and small amount of memory and are typically used to store and protect encryption keys. Cards typically need to be unlocked by a user passcode before any operations can be performed, and can be configured to self-destruct after a specified number of failed authentication attempts. Smart cards typically provide no access to the keys they store. Encryption, decryption, and signing of data is performed on the card and the result returned to the application. These cards can provide a very high level of security and meet our requirement of a key stored separately from the data it is used to protect. The challenge is one of usability with mobile devices, however. Smart card readers for smartphones and tablets do exist. However, the form factor has often significantly increased the size of a device and blocks access to the device's charging and sync port [6]. One promising solution to this is accessing a smart card via near field communication (NFC). However, NFC is not yet available on all mobile devices [7].

2.2 Mobile Biometrics

Biometric authentication on mobile devices—particularly fingerprint-based authentication—has been available for quite some time. Apple's Touch ID system on the iPhone and iPad has popularized the use of fingerprints for mobile-device unlock [8]. EyeVerify offers a mobile solution for authentication that uses the device camera to capture images of the whites of a user's eyes [9]. Although convenient, however, biometric authentication does not provide adequate security in all situations. As mentioned earlier, to provide real security, the encryption key used to protect stored content on a device must be derived from data not stored on that device. Reliably deriving a key with sufficient entropy from biometric features is an open problem, particularly with the resolution of biometric sensors currently available on mobile devices [10]. Touch ID attempts to address this problem by storing fingerprint data in such a way that it can only be accessed by the Secure Enclave, a section of Apple's processor supposedly inaccessible to the rest of the processor and that has access to a hardware UID key inaccessible to the rest of the processor [2]. Touch ID stores the actual key used to encrypt data in the Secure Enclave and releases it only after a successful fingerprint authentication. Even in this situation, however, a well-funded attacker can potentially recover device-specific keys and might not even need to if a boot exploit is found for a device such that timeout and device-wipe restrictions can be bypassed when brute forcing a passcode. (TouchID requires the use of a passcode or password as an alternate means of authentication and for authentication after a reboot. [1]). Another concern is that even if biometric authentication can be cryptographically secure, it leaves no room for plausible deniability in cases of political activism, when lives might be at risk.

2.3 Mobile Authentication Usability Concerns

Studies have shown that all-or-nothing access to mobile devices does not necessarily fit user preferences [11]. Even if FDE can be made adequately secure with sufficiently complex passcodes, it requires an all-or-nothing approach to device access, requiring authentication even for access to data and apps not considered sensitive. This is especially a concern in Bring Your Own Device (BYOD) scenarios, in which employees provide their own mobile device for use in both personal and work tasks. Users might not be open to a full device lock requiring authentication with each use [11].

3. Cross-Device Authentication and Key Exchange

The goal of this work was to create a cryptographically secure system for data-at-rest protection and offline authentication between mobile devices. Our system is based on the following design principles:

- The system should make use of hardware a user already carries and not require a new specialized device.
- The system should work on all of the most common mobile platforms, and across devices of different platforms.
- The system should be capable of offline authentication so that data remains available in situations without network connectivity.
- The authentication process should be observable by a third party without compromising the security of the system and the data.
- Neither the device with data to be protected nor the device used as an authenticator can be assumed to be physically secure. Anyone in physical possession of a device is assumed to be capable of reading the (encrypted) data on it.

Our system makes use of two mobile devices: one containing sensitive data to be protected (the *data device*), and one used to authenticate to the data device (the *authenticator device*). Data-at-rest on the data device is stored in an encrypted state at the application layer regardless of whether FDE is enabled for the device at the file system or hardware level. Files are encrypted using keys protected by a key known only to the authenticator device, which we call the *master key* K_M. K_M is available to the data device only after the user unlocks the authenticator device, generates a QR code containing a time-based encrypted version of K_M, and uses the data device to scan that QR code. K_M is

completely unrelated to any user passwords, user-generated data, device data, or hardware-based keys. In the following sections we provide implementation details for both devices, the setup and pairing process, and the authentication process.

3.1 Authenticator Device

One or more authenticator devices are associated with a data device. Each authenticator device possesses an encrypted version of K_M for which it does not have the decryption key. K_M is encrypted using an asymmetric RSA key whose private key is known only to the data device. If we denote encryption with the RSA algorithm as

$$E_{RSA}(D,K)$$

where D is the data to be encrypted and K is the key used for encryption, the authenticator device possesses only $E_{RSA}(K_M, K_{D_{public}})$. If this device were compromised, no knowledge of K_M would be revealed. If both the authenticator device and the data device were compromised, however, an attacker could obtain $K_{D_{private}}$ from the data device and then obtain

$$K_M = E_{RSA}(E_{RSA}(K_M, K_{D_{public}}), K_{D_{private}}).$$

For this reason we further protect $E_{RSA}(K_M, K_{D_{public}})$ by encrypting it prior to storage on disk with a symmetric key using the AES algorithm. This key is derived from a user passcode via the PBKDF2 algorithm for a passcode used solely to unlock the authenticator app on the authenticator device. This final layer of protection for K_M ensures that even if both the data device and the authenticator device are compromised, the attacker must still brute force the user passcode for the authenticator app to obtain the protected data. In this situation, the security of our system is equivalent to that provided by FDE.

3.2 Key Sharing

After the user has unlocked the authenticator app via passcode, the app calculates the symmetric key from the passcode via PBKDF2 and uses this to decrypt the stored value of $E_{RSA}(K_M, K_{D_{public}})$. It then must share this encrypted key with the data device so that the data that device stores can be decrypted. We chose QR codes as the means of key exchange for several reasons. First, many users are already familiar with these codes from advertisements and have scanned one before. If not, most users are still familiar with the operation of the camera on their device. Second, QR codes enabled us to easily create a platform-independent solution. Libraries already exist for QR code generation and scanning on all the major mobile platforms, and the majority of mobile devices today have at least one built-in camera. Although NFC might be an attractive option for key exchange in the future [7], it is not a ubiquitous feature on mobile devices today. Key exchange over Bluetooth might be a viable option, but it is not one we have yet explored.

The amount of data that a QR code holds varies depending on the code version and the level of error correction used. Our system uses the code to transmit the encryption key to the data device, so codes need to hold at least as many bits as the chosen key size. For our system, 256-bit keys symmetric keys protected by a 2,048-bit RSA key were used, so the code must hold at least the 256-byte block size used by the RSA algorithm with this key size. Version 17 codes, which are 85x85 blocks, have sufficient data capacity at any of the four levels of error correction defined by the standard to hold the 256-bytes needed. An example of one of these codes is shown in Figure 1.

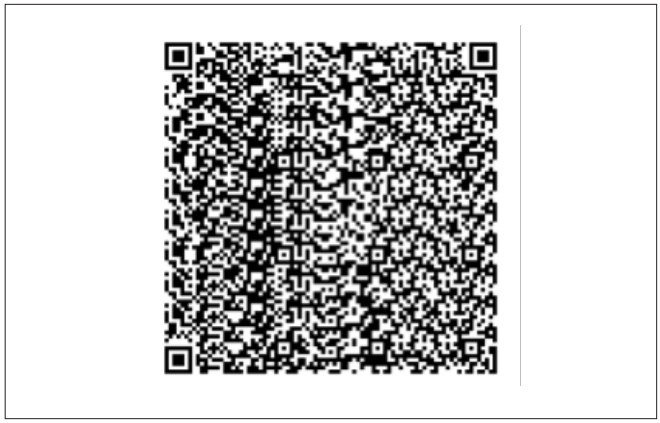


Figure 1. An 85x85 QR code with high error correction containing a link to www.air-watch.com

Because one of our design goals was to allow a third party to observe the authentication process between the data device and the authenticator device, the system needs to be immune to a replay attack whereby an attacker captures the displayed code and then later steals the data device and uses this captured code to authenticate. For this reason we further protect $E_{RSA}(K_M, K_{D_{public}})$ during authentication with a TOTP based on RFC 6238 [5]. The TOTP, which is used as a symmetric encryption key to encrypt $E_{RSA}(K_M, K_{D_{public}})$ before embedding it in a QR code, is calculated as

$$TOTP = HMAC_{SHA256}(T,S)$$

where T is the current time, defined as $T=\frac{T_{current}-T_0}{X}$, X is the step size in seconds, $T_{current}$ is the current time in seconds, and T_0 is a chosen initial reference time known to both the data and authenticator devices. S is a shared secret value known to both the data and authenticator devices. Thus, if we denote encryption with AES as $E_{AES}(D,K)$ where D is the data to be encrypted and K is the key, the value embedded in the QR code displayed by the authenticator device is calculated as $E_{AES}(E_{RSA}(K_M, K_{D_{public}}), TOTP)$.

We chose a value of 15 seconds for the time step, so a new TOTP and associated encrypted QR code is generated and displayed by the authenticator app at this interval. Longer or shorter intervals can be chosen as desired. (Shorter intervals open the possibility of authentication failures due to clock-synchronization issues between the devices, and longer intervals potentially reduce security by lengthening the window an attacker has to capture the code while observing authentication and then physically obtain the data device.)

3.3 Data Device

The data device is used to execute an application for storing and viewing sensitive content. This application stores all sensitive content in an encrypted state on disk, regardless of whether FDE is enabled for the device. Content is encrypted using the AES algorithm and, to support key rotation, an independent randomly generated symmetric key for each file requiring protection. The collection of these keys, which we refer to as the *keybag*, is then encrypted using AES with K_M as the encryption key. This hierarchical scheme is similar to the one implemented for FDE by several mobile manufacturers [2].

When the user executes the application and requests to view sensitive content, a prompt is shown requiring authentication with the authenticator device. After a QR code is scanned containing

the current TOTP-based decryption key, the application calculates K_M by first calculating the current expected TOTP, and then using this to decrypt the value stored in the QR code as

$$K_M = E_{RSA}(E^{-1}_{AES}(E_{AES}(E_{RSA}(K_M, K_{D_{public}}), TOTP), TOTP), K_{D_{private}}).$$

Where $(E_{AES}(E_{RSA}(K_M, K_{D_{public}}), TOTP)$ is the value that was scanned from the QR code and $E^{-1}_{AES}(D,K)$ denotes decryption with the AES algorithm. It then uses K_M to decrypt the keybag.

3.4 Device Pairing

Our system relies on three pieces of data shared between the data device and the authenticator device:

- S, the shared secret used for TOTP generation
- T_0, the initial time value used in calculating the current time for TOTP generation
- $E_{RSA}(K_M, K_{D_{public}})$

To facilitate this, we rely on a third-party server for managing data and authenticator device pairing establishment. After pairing is complete, authentication can occur completely offline. Pairing occurs via the following process, starting with the data device.

Data device:

1. User authenticates to the pairing server via username/password or other technique on data-containing device.
2. Data device generates an asymmetric key pair and sends the public portion of the key to the pairing server.
3. Pairing server generates a master symmetric key and encrypts it with the data device public key. Pairing server escrows the master symmetric key in case all authenticator devices are lost or inoperable, or for adding additional authenticator devices in the future.
4. Pairing server sends encrypted master symmetric key to data device.
5. Data device decrypts key and uses it to encrypt the keybag containing symmetric keys for all protected files.
6. Data device purges master key from memory. Subsequent decryption requires interaction with an authenticator device.

Authenticator device:

1. User authenticates to the pairing server via username/password or other technique on authenticator device.
2. Authenticator requests a PIN or passcode from the user for future use in unlocking the authenticator app.
3. PIN/passcode is immediately used to generate a symmetric encryption key via PBKDF2. The PIN/passcode can be salted with a device-specific attribute for added security.
4. Authenticator generates a shared secret, S, and chooses a value for T_0, then sends these to the pairing server along with a device identifier.
5. Authenticator requests the encrypted master key for the data device from the server.
6. Authenticator immediately encrypts the encrypted data device master key with the PBKDF2 PIN/passcode-derived key, stores the doubly encrypted key to disk, and purges the PBKDF2 derived key from memory.

After the authenticator device has generated a shared secret and chosen a value for T_0, and sent these and its device identifier to the pairing server, it needs to be registered with the data device:

1. User authenticates to the pairing server via username/password or other technique on data-containing device.
2. Data device requests a list of registered authenticator devices and obtains their (S, T_0, D_{UID}) tuples, where D_{UID} is a unique device identifier.

This also enables each authenticator device to have a unique shared secret so that individual devices can be revoked as needed. Registration can be performed periodically so that existing authenticator devices can be revoked and new ones added. If the need for rotating the master key K_M arises, this can be initiated via the pairing server and would require reregistering all authentication devices.

4. System Weaknesses and Future Enhancements

Although we believe our system adds significant security to data-at-rest stored on a mobile device beyond that provided by existing mobile FDE schemes, it is not unbreakable. The most obvious weakness is that if both devices are physically compromised, the attacker can potentially obtain the master encryption key by brute forcing the passcode used on the authenticator application. This can be addressed by requiring a strong password on the authenticator application, which might be acceptable from a usability standpoint because it is only needed to access specific sensitive data and not the entire device. Another potential weakness is that if the data device is compromised or jailbroken, a malicious application could run in the background and obtain the encryption key or even the data itself from memory. Many security-conscious mobile applications perform various checks on a mobile device to attempt to detect a jailbreak, but detection cannot be guaranteed. The final and potentially most feasible attack we are aware of could occur if an attacker captures a QR authentication code and then later physically obtains the data device. At this point, because the shared secret and T_0 are both known to the data device, the attacker could obtain these and decrypt the QR code. As protection from this situation, an enhancement to this system is periodic key rotation, whereby a new K_M is generated at specified intervals and shared via the pairing server with the data and authenticator devices, and the data device keybag is reencrypted with this new key.

5. Conclusion

In this work, we presented a system for enhancing the protection of data-at-rest stored on a mobile device. Although current encryption schemes provide adequate protection for many situations, ultimately a motivated attacker with sufficient resources is capable of breaking a system in which weak passcodes are used, as they often are for usability reasons on mobile devices. Our system enhances protection by protecting sensitive content with a strong randomly generated encryption key not based on a password, and using a secondary device already carried by the user to store that key. Although it is not a perfect solution, we believe our system does provide added protection over existing methods and is a good balance of security, usability, and efficiency.

References

1. Dino A. Dai Zovi. Apple iOS 4 Security Evaluation. https://www.trailofbits.com/resources/ios4_security_evaluation_slides.pdf
2. Apple Inc. (2014, October) iOS Security. https://www.apple.com/privacy/docs/iOS_Security_Guide_Oct_2014.pdf
3. EMC. RSA SecurID Hardware Authenticators. http://www.emc.com/security/rsa-securid/rsa-securid-hardware-authenticators.htm
4. D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. (2005, December) HOTP: An HMAC-Based One-Time Password Algorithm. http://www.ietf.org/rfc/rfc4226.txt
5. D. M'Raihi, S. Machani, M. Pei, and J. Rydell. (2011, May) TOTP: Time-Based One-Time Password Algorithm. Online]. http://tools.ietf.org/html/rfc6238
6. Precise Biometrics AB. Precise Biometrics AB. http://www.precisebiometrics.com/mobile-smart-card-readers
7. Yubico. YubiKey Hardware. https://www.yubico.com/products/yubikey-hardware/
8. Apple Inc. iPhone 6 Touch ID. https://www.apple.com/iphone-6/touch-id/
9. EyeVerify Inc. EyeVerify. www.eyeverify.com
10. Lucas Ballard, Seny Kamara, and Michael K. Reiter, "The Practical Subtleties of Biometric Key Generation," in 17th USENIX Security Symposium, 2008.
11. Eiji Hayashi, Orlana Riva, Karin Strauss, A.J. Bernheim Brush, and Stuart Schechter, "Goldilocks and the Two Mobile Devices: Going Beyond All-Or-Nothing Access to a Device's Applications," in Symposium on Usable Privacy and Security (SOUPS), Washington, D.C., 2012.
12. Denso-Wave. Information Capacity and Versions of the QR-Code. http://www.qrcode.com/en/about/version.html

Just-in-Time Desktops and the Evolution of VDI

Daniel Beveridge
Sr. End User Computing Architect, Office of CTO,
VMware, Inc.
dbeveridge@vmware.com

Abstract

Virtual desktops continue to be an important part of the IT landscape, delivering traditional Windows experiences to users from their local place of employment or remotely across wide-area networks to a range of endpoint access devices. From its early days, virtual desktop infrastructure (VDI) has struggled to find the right mix of cost, flexibility, and efficiency. This paper looks at the evolution of the desktop and various design strategies of the past aimed at efficiency, and it discusses new advances in VDI design based on disruptive new hypervisor technologies.

We conclude with a discussion of what VMware calls the Just-in-Time Desktop—a new provisioning and resource-management strategy that makes significant gains in areas that are traditional pain points for VDI.

Keywords: virtual desktop, provisioning, efficient design, Just-in-Time Desktop, Project Fargo

1. Introduction

VDI enables users to access desktop operating systems located remotely in a data center and executed atop a hypervisor platform. An image of the screen is send over the network from the data center to the user using a "display protocol" that transmits a compressed form of the remote screen image to the user.

Like physical desktops, virtual desktops consume compute resources, including CPU cycles, memory allocations, and storage resources such as I/O and disk capacity. Typically a cluster of hypervisors, such as the VMware® ESXi™ platform, delivers these resources to the virtual desktops they host, sharing underlying physical resources across a multitude of desktop virtual machines (VMs).

Businesses looking to deploy VDI have several key concerns as they consider deployment scenarios:

- Cost of acquisition (CapEx)
  - Hardware costs
  - Software licensing costs
- Operational costs (OpEx)
  - Ways to improve density
  - Streamlining of support and maintenance processes
  - Reduction of support tickets and security threats
- User experience: Users must have a good experience that aids productivity and meets their expectations.

2. VDI Evolution

2.1 Early-Phase Challenges

During the early phase of VDI designs, circa 2006, the technology focused on simply providing the basic desktop-in-the-data-center experience to users. The default "full clone" VM was the standard deployment method. The fledgling technology struggled to provide an adequate user experience, working hard to refine the display protocols involved and selling into niche markets in which high cost of acquisition was not a block to adoption.

As demand for the benefits of VDI grew, the cost model was given much closer scrutiny. VMware and competing vendors sought to expand the market for VDI, even working to position it as a mainstream desktop strategy.

Storage capacity was quickly identified as the biggest culprit in VDI's cost model. In the early years of VDI (2006–2008), VDI was almost always on SAN storage, which came at costs typically as high as USD $10 per GB. In some more-extreme examples, IT groups would "charge back" their costs at much higher rates to cover the overhead of SAN/NAS administration. For example, a major financial company I worked with was charging about USD $15,000 for a 300GB LUN to internal groups looking for storage support. At these prices, VDI pilots and larger deployment were hard to justify.

2.2 Reducing Storage Costs

Technology called *linked-clones* emerged during this phase to reduce the total capacity needed per desktop from approximately 30GB for the base OS down to perhaps 5GB per user. Linked-clones enable sharing of a *parent image* containing a reference copy of the base desktop OS. As VDI users make changes to their desktops, they need to overwrite blocks from the parent image; this is supported by a *copy-on-write* (COW) operation, in which the block from the parent image is copied to a journal file and portions of it are changed as necessary. The user's VM sees a logical overlay view of all the user's (COW) blocks superimposed onto the parent image. The net result is a small journal of COW blocks for each user that grows over time in response to user activity. Linked-clones were the first move in the direction of more-efficient zero-copy–based sharing for VDI—a hint of things to come.

This elegant resource-sharing solution helped reduce a primary cost element but also contained within it the seeds of a new dilemma. The ability to use less capacity was a big win but implied that fewer hard disk drives (HDDs) would be involved in supporting the VDI users. In short order, it became apparent that the next resource scarcity

was in the area of I/O operations per second (IOPS) availability. IOPS is a measure of how many discrete transactions the underlying storage system can process in parallel. VDI users often exert large surges of I/O, resulting in sudden spikes that far exceed the ability of standard HDD-based SAN/NAS systems. Such spikes result in congestion that injects latency into the user experience and in some cases result in dramatic degradation and even outages.

2.3 User Experience and I/O Quality

The emergence of flash storage in its many variations is having a profoundly positive impact on this IOPS-scarcity challenge. Affordable storage systems based on flash or hybrid SSD/HDD now offer attractive price points and help mitigate the more severe user-experience risks faced by HDD storage solutions. However, even as the IOPS-scarcity problem is mitigated, the I/O-quality issue is gaining new importance. Today's users regularly interact with their tablets and smartphones, which are wholly flash-based and to which they have exclusive access. Interactions with these devices are uniformly snappy and have raised user expectations of what a "normal" desktop interaction should feel like. Erratic I/O latencies on VDI storage generate major fluctuations in completion times for common user-driven tasks.

There is now a better understanding of the importance of uniform I/O latency for a quality VDI experience. New low-latency flash is emerging, such as peripheral component interconnect (PCI) and now Memory Channel Storage (MCS), which bring flash closer to the CPU—even onto the system's memory bus in the case of MCS. Paired with new converged storage systems such as VMware® Virtual SAN™, I/O is becoming both more scalable and higher-quality than ever before.

2.4 Rising Cloud Pressures

Even as VDI's storage woes seem to be behind it, new pressures for overall data center efficiency have emerged, driven by the need to produce low-cost desktops in the Cloud. Desktop as a service (DaaS) is a new mode of consumption wherein the virtual desktop is hosted in the cloud, usually outsourced to a cloud hosting provider. In this context, the service provider has increasing competitive pressures to reduce the cost of hosting so it can compete on price. In the enterprise context in which VDI was born and has matured, competitive pressures weren't nearly so severe—higher costs and lower utilization levels could be acceptable based on the use case. In today's emerging DaaS context, the total resource-consumption picture across CPU, memory, and I/O is receiving closer scrutiny than ever before as VDI evolves toward greater cloud efficiency.

2.5 Resource Efficiency Reboot

As VDI embarks on this more Darwinian DaaS journey, it is imperative that it seeks to achieve new levels of resource efficiency. As we've seen, VDI has already made strides with sharing beyond the basic hypervisor capabilities, leveraging linked-clones, flash technologies, and converged storage systems to overcome storage costs. However, there are still many ways in which VDI desktops continue to operate as though they were distributed PCs that happen to be located in the data center, ignoring obvious and compelling resource-sharing opportunities.

Distributed PCs have an underlying assumption that is baked thoroughly into their operational models, management tools, and OSs: PCs organize themselves as though they can be "offline" at any time and as though they are connected to one another by way of a network link of modest bandwidth. These assumption are false in the case of VDI, in which VMs are always connected and typically on very-high-bandwidth interconnects. Software vendors have been slow to exploit these fundamental differences, however, leading to VDI tools that impose inefficient methods for the delivery of application content, patching, and general life-cycle management.

3. Next-Generation Desktops: Cloud Rethink

For VDI, our working assumptions need to change. It should be assumed that the OS of each desktop has a high-bandwidth and low-latency path toward any data needed to generate and represent the desktop experience of a given user. This shift in assumptions leads to a new data center strategy wherein less work is done up front moving bits around the data center in and out of VDI VMs. Instead, we work to represent information in a semantically correct way to the OS while avoiding its actual transfer until we know with certainty that consumption will occur.

One can think of this shift in architecture as moving from a prepay system to a pay-per-use system. Sometimes a consumer is rewarded for prepaying with a lower cost per use, but in VDI this is seldom the case. In fact the pay-per-use strategy imposes only a tiny extra overhead per use while avoiding many spurious resource "charges" incurred by managing VDI as a "PC in the data center." In a typical VDI session, we don't know in advance which user activity will drive resource consumption, so the aggressive prestaging of contents used by PC-management tools is largely wasted. The VMware View™ Persona Management™ product took steps in the zero-copy direction, representing Windows profile contents to the OS without a full roaming profile transfer. However, the method used still generated lots of I/O in support of writing small "stub" files that must be laid down to trick Windows into believing that the full profile is present.

Even with the latest flash storage technology, today's VDI provisioning is time-consuming and very CPU-intensive, involving multiple reboots, I/O storms, and CPU saturation on the hypervisor during mass provisioning. Tomorrow's VDI must do better, conserving precious shared resources more aggressively than ever before by leveraging the close proximity of information building blocks to effect a zero-copy architecture.

3.1 Zero-Copy Architecture

For desktops to work properly, the OS must believe that all of its resources are available and ready for consumption, yet underneath the various abstractions, the OS kernel has no way to know if the actual physical resource is present beneath the semantically correct veneer. *Zero-copy architecture* is a design strategy wherein information is represented to the OS by means of appropriate metadata and pointers so that all content is believed to be present, while ensuring that content is only transferred at the point of consumption for a pay-per-use model (see Figure 1). Zero-copy

design creates for the OS a kind of holographic representation of its underlying resources that dynamically transforms into reality whenever the OS touches any part of the holograph.

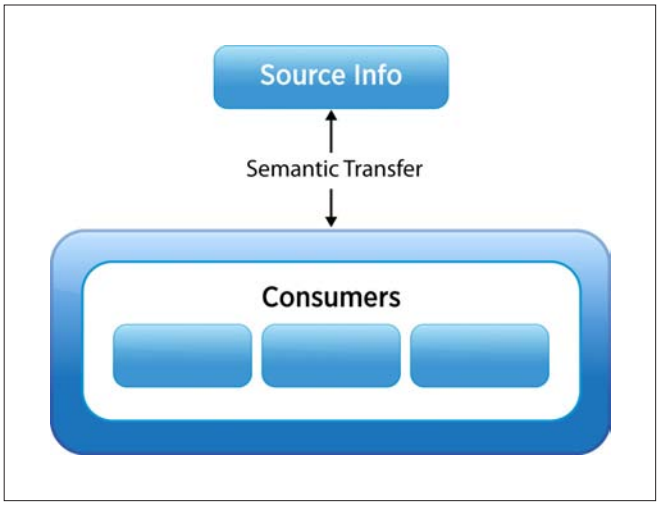


Figure 1. Zero-Copy Architecture

3.2 Zero-Copy for Memory: Project Fargo

Zero-copy methods have been used in computing for some time[1], chiefly to offload CPU from simplistic but intensive tasks. Hardware based examples of zero-copy include DMA, memory mapping using the MMU, and more recently, CPU-to-GPU offload[2]. Software use of zero-copy methods has generally focused on efficient file-system data movement with examples in Unix, Linux[3], Windows and Java. [4] In networking, RDMA[5] and VMware NSX[6] apply zero-copy to networking, moving data with minimal CPU involvement in RDMA and minimal upstream router involvement in the case of VMware NSX. Comprehensive attempts to apply zero-copy architecture are rare but do exist, notably in extreme performance environments such as data gathering for particle physics experiments at CERN[7].

In the hypervisor context, work has been done on various zero-copy networking[8] and inter-VM communication mechanisms.[9] However, less precedent exists for end to end zero-copy architecture encompassing hypervisor and guest OS. This paper discusses ways the hypervisor and guest OS software elements can work together to represent information to desktop applications, avoiding transmission entirely instead of simply making transmission more efficient. Avoidance of transmission is achieved by retaining information sharing until consumption forces replication or transmission of the shared resource. Various methods are employed to project the shared information into the OS using only minimal metadata operations, producing the illusion of dedicated resources to the OS without the usual overheads of creating separate instances.

Applying zero-copy design principles to hypervisor memory state, we want to avoid the copying of memory pages inside a hypervisor host until we know for sure that they cannot remain in a shared state. PC design assumes that each machine is standalone and must be booted up to create its initial memory state. In the hypervisor, however, we have an opportunity to start with a memory state that represents a logical starting point for the desktop—storing this only once and using memory pointers toward this content for new desktops on the same hypervisor host.

Originally described in [10] is a new method of VM cloning that involves creating a parent VM booted and prepared to a desired state. This parent is then stunned on the hypervisor, no longer scheduled, yet its memory contents remain in physical RAM. VMware has implemented this "VMFork" method in its ESXi hypervisor in a technology project we named Project Fargo (see Figure 2). Initially used for Linux VMs, this new provisioning approach is both high-speed and resource-efficient, skipping the CPU and I/O cycles normally consumed during a boot of the OS, and completing cloning operations in less than a second.

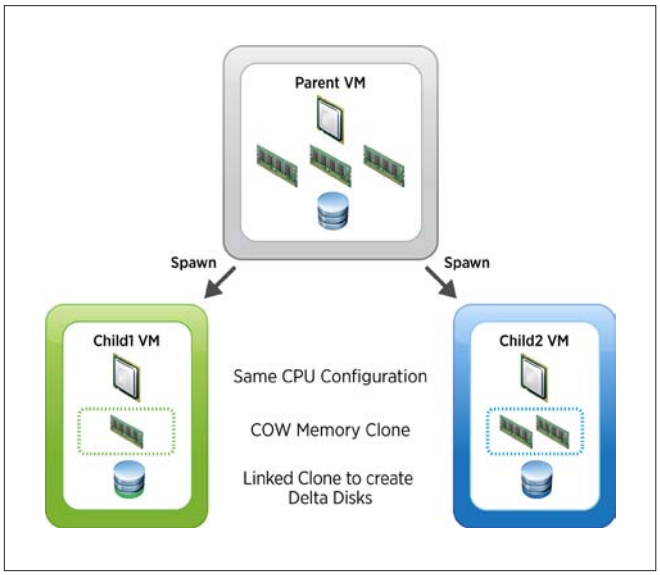


Figure 2. Project Fargo VM Cloning

3.3 The Customization Challenge

Project Fargo initially creates a child VM that is in the identical state as its parent and resumes execution at exactly the point at which its parent was stunned. A series of housekeeping steps are needed on the child VM to ensure proper functionality. For example, the network driver is cycled to force initialization with its own MAC address and IP address. Other steps might be needed, depending on the OS, to establish a unique machine name or network identity. This process is known as *customization*.

In the case of Windows, customization poses some special challenges because the key activities performed during the standard Microsoft Windows Sysprep tool or even the VMware QuickPrep tool require one or more reboots upon completion. However, for VM forking to retain its memory sharing and other resource benefits, the reboot must be avoided. Thus, to exploit Project Fargo for Windows, a new customization technology was developed that performs the key customization tasks—such as machine-name change, Active Directory join, and even Key Management Service (KMS) volume license activation—without a reboot.

VMware has developed the necessary customization technology to enable the forking of Windows VMs with customization completing in roughly five seconds after the fork event. Windows desktops can now be created using Project Fargo in tandem with the new VMware Quicker Prep customization technology (see Figure 3). This combination results in memory-state management moving to a zero-copy architecture, wherein all memory pages are initially shared with pointers in the hypervisor, and only as the OS actually needs to overwrite a given page is a copy made and modified. Memory pages can now be treated in much the same way that linked-clones handle blocks on disks: All contents are initially shared, and a change journal is generated per VM that contains unique state for the resource. This stands in stark contrast to VDI 1.0 designs that create a new set of unique memory pages for each desktop that boots or reboots.

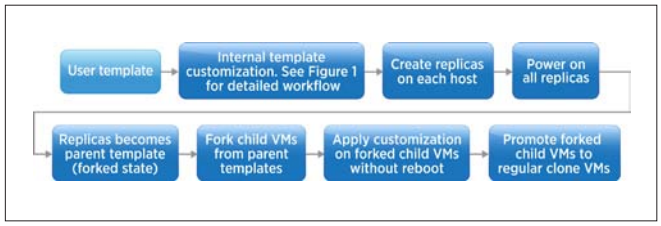


Figure 3. Quicker Prep Customization

3.4 Application Management

When it comes to desktop VMs, one size doesn't fit all. A company can have many departments, each with its own application requirements. Users in different countries or business units will need different suites of application for their daily work. To take advantage of zero-copy desktop provisioning using Project Fargo, an image of the parent VM must stay resident in the memory space of each hypervisor host. But which applications would we place into this Project Fargo parent VM, given the diversity of user groups? It might be possible to have multiple parents per hypervisor host, but with desktops using 2 to 4GB of memory, creating a parent VM for each department could lead to a huge memory overhead, undermining the efficiency goals of the architecture. Alternatively, we could choose to keep the parent VM as generic as possible, including only applications used by everyone. We would then be able to have just one parent VM per hypervisor but would need to find a method to quickly customize child VMs for the specific needs of their target users.

3.5 VMware App Volumes: Zero-Copy Delivery for Applications

Traditional application frameworks treat all desktops as though they were PCs located in the data center. Information is moved in and out of VMs through the network just as would be done among distributed PCs. Application packages are usually pushed into the OS in advance of consumption, though some streaming approaches strike a middle ground by transferring only an initial set of data and streaming the rest on demand. Even these partial delivery models often impose some form of caching on the desktop to account for it going offline—a practice that doesn't make sense in the data center.

The hypervisor is in the unique position of logically surrounding the desktop, with the ability to present vast amounts of information almost instantly to the VM using an obvious but underexploited method. VMs are constructed with virtual disks, and the hypervisor can mount a virtual disk any time. Just like insertion of a USB drive that might have hundreds of gigabytes of data, a virtual disk mount presents the file system metadata to the OS almost instantly.

VMware App Volumes™ leverages this highly efficient means of representation to deliver application contents to the OS on the fly without needing to make any copies of actual contents in the desktop OS. App Volumes uses filter drivers to virtualize the Windows Registry and file system so that contents appearing on the mounted disk can be logically merged into the OS at their proper locations—as though they were actually present on the C: drive or Windows Registry. A mapping file on the mounted virtual disk shows where the contents should be represented in the OS, but no data transfers actually take place until the user launches or otherwise consumes information on the mounted virtual disk.

The virtual disks can contain one or more applications and are mounted to the target VM in read-only mode, which enables them to be mounted to many VMs simultaneously. Each department can have its own virtual disk containing key applications that will be mounted to the desktop at user login based on user identity. Users can also be given an individual read-write virtual disk onto which their own custom application installs will be directed. This user disk will roam with the user, mounting along with their IT-provisioned apps to whichever VM the user next uses upon logging in to the VDI broker. Figure 4 shows a traditional "PC in the data center" versus an App Volumes–based VM with its plug-in applications.

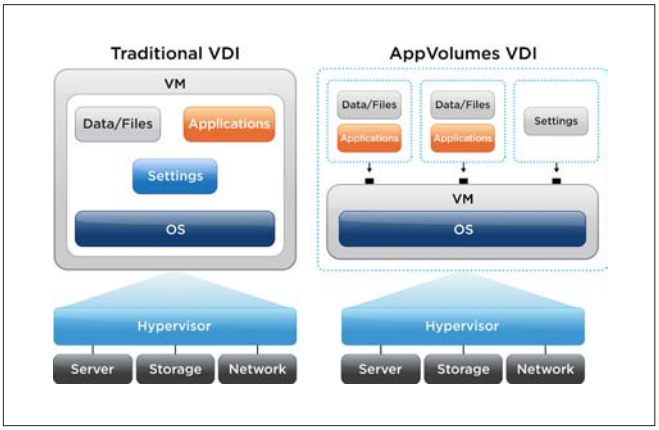


Figure 4. Zero-Copy Application Delivery: App Volumes

VMware layering technology extends the zero-copy architecture beyond what an App Volumes virtual disk itself achieves. Using the hypervisor platform to create an abstraction of the virtual machine disk (VMDK) itself, a synthetic block device (SBD) presents a file system whose contents are generated based on a backing repository yet delivered on a just-in-time basis as the OS makes queries to underlying disk. The SBD allows representation of information in block format based on information that resides elsewhere, such as a VMware Mirage™ single-instance store, which contains application layers stored as file objects. Instead of copying such file objects into a VDI desktop as traditional PC-management tools would do, the SBD can represent an application as a virtual disk under the VM—a first zero-copy step such that our App Volumes technology can, in a

second zero-copy step, merge it into the OS in the correct locations; this two-step zero-copy representation enables projection of information up to the OS in a semantically accurate way based on assets that are stored outside the VM and in different nonnative formats. Figure 5 shows a full zero-copy provisioning cycle leveraging Project Fargo for zero-copy memory-page management together with SBD + App Volumes for application representation to the OS.

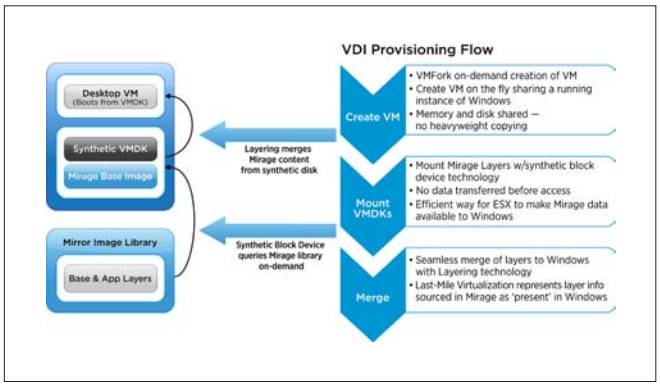


Figure 5. Synthetic Block Device + App Volumes at work

4. Project Meteor: Converging Zero-Copy Technologies

Pairing Project Fargo for machine provisioning and the App Volumes product for application insertion leads to a fully zero-copy provisioning process wherein a desktop can be created and customized to the needs of a specific user in seconds without reboots or heavy data transfers. This leads to a breakthrough in overall speed of provisioning to the point where it is no longer necessary to precreate desktop VMs. They can now be created at the point of request as the user logs in to the VDI broker, giving rise to the Just-in-Time Desktop (JIT desktop). At VMworld 2014, VMware announced Project Meteor, which aims to bring together the multiple zero-copy technologies needed to realize the JIT desktop.[11]

4.1 The JIT Desktop

This paper has examined some of the prior attempts to achieve data center efficiency in the evolution of VDI. Although progress was made in storage efficiency, the consumption patterns for CPU and memory management largely mirrored the practices and costs of distributed PCs, requiring intensive and time-consuming mass-provisioning operations. Never before has it been possible to create a fully customized desktop on demand in seconds. Prior strategies might have achieved some degree of personalization on demand but failed to accommodate user-installed applications. Combining zero-copy machine creation with Project Fargo and zero-copy application provisioning with App Volumes + SBD enables the full construction of the desktop in seconds, ushering in the age of the JIT desktop.

JIT desktops borrow their name from manufacturing innovations that strive to reduce the cost of inventory by creating product rapidly when orders arrive. The term is also known in the context of compilers that retain the speed of compiled code but amortize the resource costs of the compile job across time by triggering small granular compilations as code executes. JIT desktops likewise seek to avoid inventories of desktops created in advance of consumption and similarly spread the greatly reduced costs of provisioning over time for reduced resource-consumption storms and surges associated with mass provisioning.

Many operational benefits can be derived from JIT desktops. The heavy resource consumption associated with creating large pools of desktops disappears due to the zero-copy resource sharing and the newfound ability to spread the remaining load over time by creating desktops only at the point of request. The complexities of image management are dramatically reduced by the ability to dynamically alter application contents without changing the central OS image and without the need for long maintenance windows. Everything becomes dynamic, rapid, and resource-efficient.

Resource storms associated with mass boot operations, "recompose events" wherein the central image must be changed, or software distributions can be eliminated by the JIT desktop. Underlying all these operational gains is the architectural principle of moving from prepay to pay-per-use in the delivery of information to the OS. By incorporating zero-copy methods of representing information into the various layers of provisioning, we move from mass manipulations of data to point manipulations of metadata. Collectively, these methods enable a tailored desktop to be created upon request in roughly five seconds, which is under the threshold at which users will perceive any meaningful delay in the servicing of a request for a desktop session.

4.2 JIT Innovations: Extending the Paradigm

Zero-copy architecture enables us to cleanly distinguish shared from unshared resources: Linked-clones on disk represent unique storage state, COW memory pages represent unique memory state, and the App Volumes user disk captures customized application state to a discrete virtual disk. The clean state separation that derives from and relies upon the zero-copy strategy enables some interesting follow-on innovations, including improved suspend/resume of desktops, more efficient "vMotion" (live movement of VMs), and better handling of machine-identity issues.

4.2.1 JIT Suspend/Resume of Desktops

In the course of a given desktop session, VDI users typically disconnect from their remote desktops and reconnect at a later time such as later in the day or the next day. This disconnect/reconnect pattern results in many VMs remaining powered on but idle on the shared infrastructure. Users dislike forced logoffs because it takes time and effort to restore their desktop state at next login. Suspending the running machine to disk has been too I/O-intensive, and resume operations have taken too long to be viable when users log back in to the broker. However, JIT desktops can safely assume that a set of shared memory pages will be retained in the hypervisor—the Project Fargo parent VM—even if a VM is suspended. Resume operations can therefore simply reattach the set of COW memory pages to the already resident parent image. The set of COW memory pages can be as little as 25% of the total memory space, and this can be compressed for further reductions allowing as little as 10% of memory state to be sent to disk during a JIT desktop suspend operation (see Figure 6). This in turn allows for very rapid resume operations that take a few seconds, fast enough to avoid aggravating users logging back in to the VDI broker to resume working on their desktops.

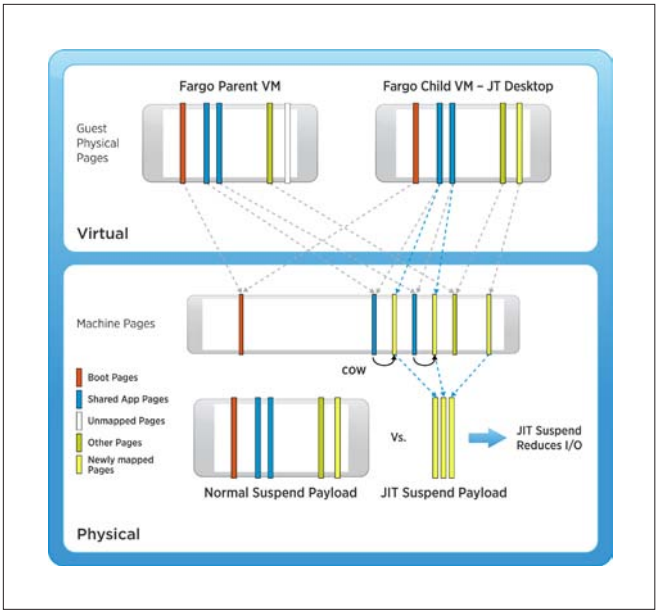


Figure 6. JIT Suspend/Resume

4.2.2 JIT "vMotion": Optimized Live Migration of VMs

Similarly, the live migration of VMs—"vMotion" operations—can take advantage of a JIT desktop's zero-copy construction by simply moving the set of COW pages, instead of the full memory space, between hypervisor hosts during a migration, logically reattaching these COW pages to an identical Project Fargo parent VM that can be transferred in advance and will therefore be already resident on the target hypervisor host (see Figure 7). The reduction in data volume of such migrations can facilitate efficient cross–data center migrations with greatly reduced WAN consumption.

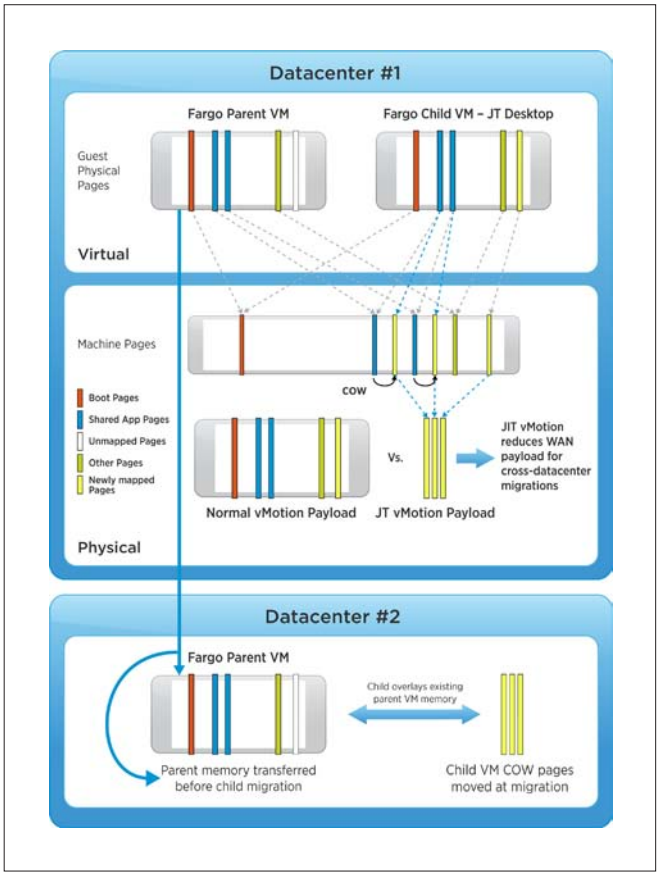


Figure 7. JIT vMotion Design

4.2.3 JIT Desktop Machine-Identity Persistence

In addition to the innovations that take advantage of zero-copy memory management, JIT desktops have a big speed advantage that allows for a tailored desktop to be assembled on the fly. In prior VDI designs, floating pools of VMs had to be created in advance of user requests due to the time involved in generating a desktop VM. This resulted in users being assigned to a random VM in the pool each time they logged into the broker and, by implication, receiving a different Windows machine name each session.

In the JIT paradigm, we assemble the personalized desktop on the fly—at the point the user makes a request to the broker. We know the user's identity at this point and can create a desktop with a stable Windows machine identity across sessions by inserting the same machine name and Active Directory account into the desktop at each session request (see Figure 8). Such machine-identity persistence across sessions is a benefit to a range of software that checks for such persistence as part of license compliance or security audit or for a variety of other reasons. With JIT desktops, machine-identity persistence enables users to install applications that persist atop an App Volumes user disk with the assurance that even software needing a persistent machine identity will function properly at the next user login.

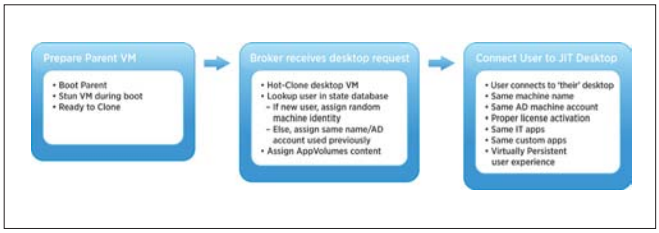


Figure 8. JIT Machine Persistence

5. Conclusion

JIT desktops are a breakthrough in VDI provisioning and management, offering IT an opportunity to deploy virtual desktops without lengthy and resource-intensive mass-provisioning operations. The speed and resource efficiency of this new zero-copy–based strategy breaks new ground by rethinking how information is used within the software-defined data center. Instead of mass manipulations of bulky data elements, JIT provisioning opts for holographic representations of information elements that collectively produce a pay-per-use resource-consumption model and exponentially faster VDI desktop assembly.

By constructing desktops in the JIT zero-copy paradigm, opportunities emerge for optimizing additional life-cycle and management activities in ways not possible within the traditional desktop paradigm. Initial and ongoing costs are reduced even while users have a flexible and high-performance desktop experience. JIT desktops set the pace for reimagining how we will provision all types of VMs and even containers in the future, ushering in an era when fewer "information calories" can be expended in pursuit of our workload provisioning and life-cycle goals.

Acknowledgments

Thanks to Gabriel Tarasuk-Levin for his work on Project Fargo within the VMware ESXi platform, to Frank (Hui) Li for his critical contributions to and collaboration with our Fargo-friendly Windows customization technology, and to Matt Conover for his insightful use of hypervisor content-delivery mechanism within our App Volumes product. Thanks also to Banit Agrawal for help in scale testing and analysis of JIT desktop execution profiles and to Darius Davis, Regis Duchesne, and Tal Zamir for their inspiring work on synthetic block devices.

References

1 Zero-Copy Architecture: http://en.wikipedia.org/wiki/Zero-copy

2 CPU-to-GPU Offload with HSA - http://en.wikipedia.org/wiki/Heterogeneous_System_Architecture

3 Zero-Copy in Linux - http://www.mirlabs.org/ijcisim/regular_papers_2011/Paper2.pdf

4 Zero-Copy in Java - http://www.cs.cornell.edu/home/chichao/tr1708.pdf

5 RDMA - http://en.wikipedia.org/wiki/Remote_direct_memory_access

6 VMware NSX - http://www.vmware.com/products/nsx

7 Zero-Copy with CERN's CMS data gathering architecture - http://cds.cern.ch/record/1458469?ln=en

8 Zero-copy transmission within the hypervisor - http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20110126195.PGNR.

9 FIDO: Zero-Copy communication between VMs - https://www.usenix.org/legacy/event/usenix09/tech/full_papers/burtsev/burtsev_html/index.html

10 H. Andres Lagar-Cavilla, Joseph Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. 3rd European Conference on Computer Systems (Eurosys), Nuremberg, Germany, April 2009. http://sysweb.cs.toronto.edu/publications/86 http://sysweb.cs.toronto.edu/snowflock

11 Kit Colbert (2014). The importance of Project Meteor & Just-in-Time Desktops. https://www.youtube.com/watch?v=TtLIMqRVwH8&feature=youtu.be

Connectivity and Collaboration in VMware vCloud Suite

Ravi Soundararajan
VMware Inc.
ravi@vmware.com

Shishir Kakaraddi*
VMware Inc. and Netflix Inc.
shishirkakaraddi@gmail.com

Abstract

Although there are a wide variety of tools for administering and monitoring virtual infrastructure, these tools often are unable to answer a simple question: When an administrator gets a phone call in the middle of the night regarding some sort of infrastructure issue, whom should she contact? The answer to this question requires a catalog of assets in an environment, an association of those assets with their owners, and a means of communicating with those owners. This information is typically distributed among each of the management tools, and no single tool has a global view that can satisfy all of these needs.

In this paper, we posit that many such questions can be solved by providing simple techniques for creating and maintaining connections among objects and also providing techniques for collaboration among human users. We make an analogy to the "Internet of things" and create an "Internet of virtualization entities." With a network of connections and a simple means of aggregating data from multiple sources, we can answer a number of common IT administration questions. As a proof of concept, we describe a preliminary prototype that is built by integrating the Socialcast® collaboration framework with a number of virtualization tools within VMware vCloud Suite®, providing a *persona*-like presence for all objects within a virtualization hierarchy. We discuss our prototype architecture and detail a number of simple IT administration use cases that employ this prototype.

Categories and Subject Descriptors: virtual machines, system management, cloud computing, graph databases, social media

General Terms: performance, management, design

Keywords: virtual machine management, cloud computing, data center management tools, collaboration

1. Introduction

The entire premise of the "Internet of things" [24] is that an increase in the number of networked information sources can lead to more agility in decision making. For example, if every car on the road has a sensor, traffic congestion can be more accurately measured, and if every widget on a factory floor has an IP address, then doing a large-scale recall on a product when a defect is detected becomes much simpler.

Many problems in IT management can benefit from connectivity similar to that of an "Internet of things." For example, a common complaint among IT professionals is that when a problem occurs, knowing whom to contact is difficult. This issue is exacerbated in a virtual environment. For example, if a virtual machine (VM) is performing poorly, then multiple administrators might need to be contacted: the VM owner, the host owner, and perhaps even the storage owner. With the seamless movement of VMs across infrastructure, identifying these owners can be difficult. Moreover, after these owners are identified, collaboration becomes the next problem. Ultimately, the ability to link users and assets and to query those assets for the appropriate data is essential for efficient functioning in a data center.

Although existing products are able to manage and monitor individual pieces of a virtualization hierarchy (e.g., indicating when a VM might be resource-starved), the wide variety and scope of such tools make it difficult to get a unified view of the entire infrastructure. For example, VMware vCenter Server™ [20] monitors hosts and VMs and can signal alerts if a particular metric has exceeded a threshold. If a user has also deployed VMware vRealize™ Operations Manager™ [17] to monitor these hosts and VMs, then the user will also get alerts from vRealize Operations Manager. In this situation, a problem that shows up as an alert in vCenter Server might not appear as a problem in vRealize Operations Manager, and even if both show the same issue, an administrator can thus get multiple notifications and multiple emails in response to a single problem. The problem can get worse as more solutions and more notification mechanisms are added to an environment.

The goal of this paper is to use improved connectivity and an enhanced aggregation of data to help solve the problem of the lack of a single view across an infrastructure. We model connectivity by using the analogy of a social network. Whereas previous work [12] has explored social media metaphors for linking users to their entities, we go a step further and link all elements of a virtual infrastructure—including hosts, VMs, users, and monitoring software—in a social network. The advantage of a social networking portal is that it is an intuitive interface that is architected for connecting entities to one another and also has a built-in collaboration framework. We demonstrate how a simple linkage of disparate data sources such as an operations-management tool (vRealize Operations Manager),

*This work was done while Shishir Kakaraddi was at VMware.

a log-analysis tool (VMware vRealize™ Log Insight™) [18], a cloud-management platform (VMware vCloud Director® [16]), and a virtualization-management tool (vCenter Server) can simplify the process of debugging and identifying stakeholders when problems arise.

The structure of this paper is as follows. In section 2, we provide further motivation of the problem statement. In section 3, we give a brief overview of our prototype. In section 4, we demonstrate how our prototype can solve various use cases in virtualization administration. In section 5, we present a brief overview of related work. We conclude and speculate on future directions in section 6.

2. Difficulties in Connecting and Collaborating

End users who try to identify stakeholders during an issue often encounter three problems: lack of unified notifications, inability to connect owners and assets, and inability to collaborate on the problem at hand. Consider a simple example in which a VM experiences poor performance because there are not enough physical CPU resources to run the VM (e.g., it has high ready time). vCenter triggers an alert if ready time exceeds a predetermined threshold, vRealize Operations Manager triggers alerts when it detects that ready time is larger than its historical average, and Log Insight may alert of a failure that is indirectly related to low CPU (e.g., perhaps a VM live migration is unable to complete because insufficient CPU is available for the operation). Thus, in this case, it is conceivable that an administrator will get three separate alerts in response to the same problem. Moreover, these types of notifications (typically using email) are inherently private to each stakeholder, so administrators might not be aware who else has been notified.

Moving beyond the issue of notifications, after it is determined that a VM is experiencing high ready time, the next step is to ask why, and this can involve finding the owner of the host that is running the VM in order to find out why the host is oversubscribed. Although finding the host running a VM can be relatively straightforward, unless the owner of the host has left some "bread crumb" (e.g., a tag on a host or a custom field [21] on the host), associating the host with its owner can be challenging. Maintaining these connections programmatically can be onerous for an end user.

After an administrator has found the proper owner, the next step is to try to resolve the problem, which requires some method of collaboration. Products such as vRealize Operations Manager, Log Insight, and vCenter Server do not have a facility to enable administrators to interact in real time. Instead, administrators must use email or utilize tags and then alert other administrators that such tags have been modified. Although one solution might be to utilize the "message of the day" feature in vSphere Web Client, this is clearly undesirable because anyone who is an administrator of vSphere sees the message, rather than just the administrator of the entity under consideration.

Whereas the scenario above is related to problem solving, another common workflow is simply to track the life cycle of a VM or a host and maintain an audit trail. For example, a VM owner might want to be notified when the host running the VM is being placed in maintenance mode, so that the VM owner can anticipate potentially slow performance. The same issues as above arise in implementing such a workflow.

Our proposal is to utilize the capabilities of social media to help alleviate the problems above. Our approach is threefold. First, we add all entities (VM, host, management solution, or user) to the social network and connect them to one another. This simplifies the process of identifying stakeholders. Second, rather than utilizing email notifications, we post notifications to the social network. By posting notifications to a central location where all owners of an asset will see the message, we mitigate somewhat the problem of not knowing who has been sent a notification. Finally, because users are able to comment on specific messages, and these comments are stored within the social network and associated with a specific asset (e.g., stored on a VM's home page), collaboration among various administrators is simplified. We expand upon prior work in applying social media tenets to virtualization management [12] by linking more entity types, creating more connections, and incorporating collaboration. This web of connections can also be seen as a concept graph [9].

3. Prototype Design

3.1 Overview

In this section, we describe a preliminary prototype design for our "Internet-of-things" management framework. Our design uses Socialcast [19] to create the social network to connect our entities, including VMware ESX® hosts, vCenter Server instances, VMs, users, and management products such as vRealize Operations Manager and Log Insight. With these connections in place, we can integrate data from multiple sources into a consolidated view that can be helpful for monitoring the various pieces of the system, archiving activities on various components, and discovering connections among components (e.g., noting if certain tasks in vCenter Server cause storms of log messages in Log Insight, or simply discovering all users who are interested in a given VM or host).

In Figure 1, we show a block diagram of our design. The heart of the system is our SCX service. SCX uses public APIs to retrieve inventory data and event data from vCenter Server, metric data from vRealize Operations Manager, and logging information from Log Insight. SCX creates Socialcast users for each host and VM within vCenter Server, as well as for vCenter Server itself. SCX also creates users for vRealize Operations Manager and Log Insight. Finally, users are created for administrators and for the end users of the VMs in the infrastructure. When the infrastructure entities are available in Socialcast, end users or administrators can register interest in these entities by following them, and they can keep track of sets of VMs by organizing the VMs into groups. In addition, other human users can also be added to these groups, providing a convenient association between resources and stakeholders. For example, imagine that there are a host of errors related to a MySQL database application, and that these are posted to the page for the MySQL database entity. To have

a MySQL database expert look at such logs, one need only tell the MySQL expert to join the appropriate group containing this application, rather than having to provide credentials for the specific database instance or the VM running the database.

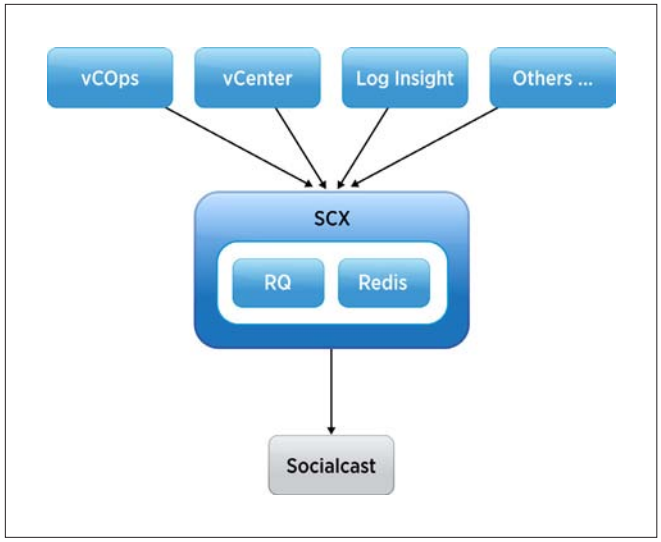


Figure 1. Integration of vCloud Suite with Socialcast via SCX. SCX receives data from various components of vCloud Suite and uses Socialcast to maintain connections and store message data. Socialcast enables collaboration among human users.

After the initial connections are established, SCX tracks changes to the vCenter Server inventory and pushes these changes to Socialcast. In addition, SCX also tracks events (e.g., errors or warnings associated with a VM, or tasks performed on a VM) within vCenter Server, vRealize Operations Manager, and Log Insight, and pushes those to Socialcast. Because SCX can retrieve inventory information from vCenter Server and connectivity information from Socialcast, it is able to enrich the content that is pushed in Socialcast. For example, SCX can take a log message from Log Insight and replace the VM identifier with the name of the VM (and can optionally provide a web client link). Because Log Insight has no knowledge of inventory structure by default, it is unable to make such an association by itself (although when configured properly, it is potentially able to determine some of these connections). Although vCenter Server does have a web-based UI from which one can view events for the VM, this same page does not store anomalies that are detected by other tools such as vRealize Operations Manager, and it also does not incorporate log messages related to the VM (which might be stored in Log Insight). Beyond just aggregation of information in a single place, when this data is available in Socialcast, all interested parties are notified that an event has occurred on this entity. These administrators can then view the Socialcast page for the VM, comment on it, and collaborate to solve the problem.

3.2 Implementation Challenges

The SCX prototype retrieves information from a variety of sources and presents the data in Socialcast. One of our challenges in designing SCX is that there are multiple sources of truth. For example, vCenter Server contains information about hosts and VMs, and this data must be mirrored in the Socialcast database. For our initial prototype, we enforce an eventually-consistent model for the vCenter Server entities that appear in Socialcast. We monitor for real-time changes in vCenter Server by using the vSphere *PropertyCollector* notification API [21], which enables a user to specify an entity (e.g., host or VM) and which properties of that entity (e.g., power state) to monitor. For other solutions without such a notification API, we use a polling scheme with a configurable 5-second polling interval. Within SCX, we store data in a Redis [8] key-value store backed by a file. As a result, if vCenter Server or Socialcast crashes and restarts, we are able to view the latest information in vCenter Server, do a diff with our stored copy, and then edit the list of users in Socialcast appropriately.

One big challenge with SCX is filtering of relevant events. An administrator following a large number of assets might have a large number of messages to examine, which would then defeat the purpose of aggregating messages in a single place and would produce more noise than useful information. Our initial solution is to focus on task events (such as reconfiguring a VM or editing the network settings on a host) as well as error events. We can then create a custom stream within Socialcast specifically for tasks and errors, as suggested in previous work [12], and we can also use hashtags to further characterize events. The popularity of a given hashtag can give insight into which conversations to follow and which entities to examine.

4. Use Cases

4.1 Slow VM

To demonstrate the value of our system, we consider a concrete use case: An application is showing degraded performance. An end user posts a message to the application group indicating that the application is slow. The VM owner is a member of the same group and navigates to the VM's page, which shows a high-ready-time alert from Log Insight as well as a "reconfigure virtual machine" task event from vCenter Server. The reconfigure event also indicates who performed the action. At this point, the VM owner is able to determine whom to contact to find out why the VM was reconfigured. In Figure 2, we show a screenshot from Socialcast that shows how SCX aggregates the information for this sort of interaction and then

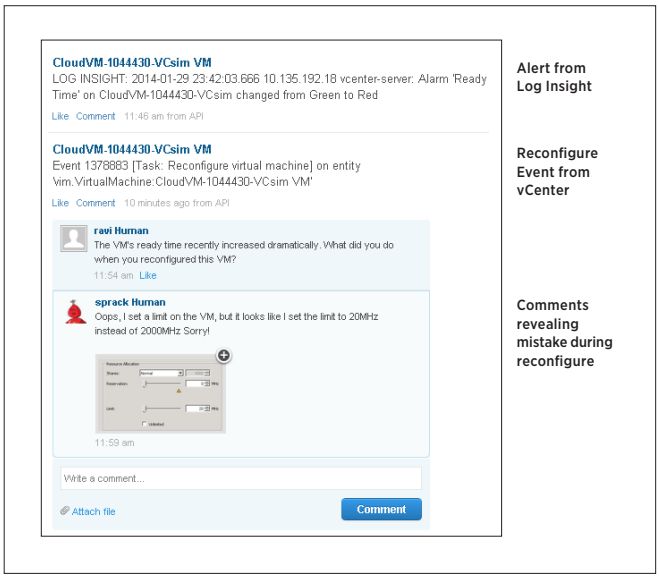


Figure 2. Slow VM. A VM owner experiences slow performance. She sees an alert from Log Insight indicating high ready time. She also notices a reconfigure task on this VM. When she asks about the reconfigure task, another administrator indicates that a mistake was made during reconfiguration and attaches a screenshot for clarity.

presents it to Socialcast. This collaborative component helps shorten root-cause analysis. Because a reconfigure event is usually not considered an anomalous event, tools that focus solely on alerts to do performance debugging might miss the root cause of the issue. As an added benefit, because Socialcast can be configured to keep messages indefinitely and such messages can be searched, an administrator can issue a search query within Socialcast to determine if anything else has happened to her VM.

4.2 Integrated Workload Profiling

We next provide an example of using connectivity to help combine workload data from disparate data sources—namely, the cloud-management layer and the virtualization layer underneath. For cloud management, we use vCloud Director [16], and for virtualization management we use vSphere (specifically, vCenter Server). The vCloud Director layer issues high-level tasks (e.g., provision a group of VMs connected to a given network that is isolated from other VMs), and this is translated to a sequence of low-level commands (e.g., create an isolated network, create a set of VMs, connect them to this network, and power them on). An administrator using vCloud Director might want to understand what load it puts on its underlying vCenter Server: A small number of vCloud Director tasks can result in multistep workflows in vCenter Server, and if the resulting load on vCenter Server is sufficiently high, this can result in slow task completion, which then surfaces to the vCloud Director administrator as a slow provisioning operation. By understanding the mapping between high-level vCloud Director tasks and low-level vCenter Server tasks, we can determine if there are optimizations in the low-level tasks that can improve the performance of the high-level tasks. We have used SCX to analyze traffic from a real-world scenario (the hands-on labs at the VMworld 2013 conference [14]) and identified optimization opportunities as a result.

vCloud Director has an API to retrieve vCloud Director tasks but lacks an API to retrieve the underlying vCenter Server tasks that result. This API is not present because vCloud Director is intended to hide such low-level details from the end user and instead simply present a "self-service portal" interface. vCloud Director, however, does have an API to determine which vCenter Server instances are connected to it. With SCX, we create a vCloud Director user in Socialcast, create users for the vCenter Server instances connected to this vCloud Director instance, and then create a group for all of the respective administrators. SCX queries vCloud Director for its list of tasks and then queries the connected vCenter Server instances for their tasks. The result is then posted on a graph in Socialcast and viewed by the vCloud Director administrator and all of the vCenter Server administrators, because they are all connected in a group. Such a graph, shown in Figure 3, indicates that for the workflows in this vCloud Director instance, each vCloud Director task corresponds to roughly four vCenter Server tasks. Because vCenter Server has limits on the number of tasks it can perform concurrently [22], by knowing how many tasks are created in vCenter Server for each task in vCloud Director, we can potentially throttle the number of tasks submitted by vCloud Director, which results in a better user experience.

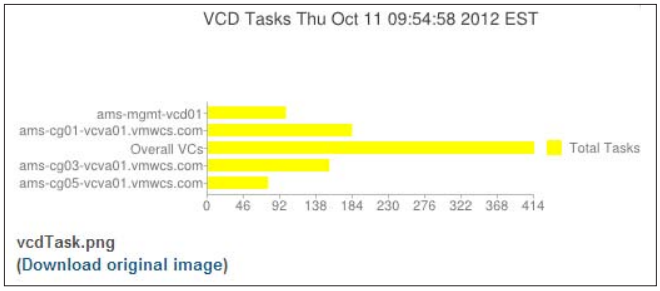


Figure 3. Mapping vCloud Director tasks to vCenter Server tasks. SCX grabs the total number of tasks performed by a vCloud Director instance (ams-mgmt-vcd01) and then grabs tasks for each of the three vCenter Server instances managed by this vCloud Director instance. The sum of vCenter Server tasks is "Overall VCs." Here we see that 100 vCloud Director tasks map to approximately 400 tasks for this workload. The chart can be viewed by any user following this vCloud Director instance or any of the associated vCenter Server instances.

4.3 vCenter Server Database Debugging

Our next example is a fairly conventional collaborative scenario involving a common customer issue: The vCenter Server database stores historical data, but as the level of detail (the so-called stats level) is increased, the volume of data persisted to the database becomes large enough that database queries can become slow. Moreover, other operations unrelated to statistics can be slowed down.

Determining why vCenter Server or the database is encountering issues requires coordination with a number of users and machines. In terms of human users, the vCenter Server owner, the database owner, and the ESX host administrators must communicate. These users, in turn, must be aware of the underlying hosts that they are managing. One way to organize these users and machines is by creating a vCenter group that consists of the vCenter Server instance, the database, the ESX hosts, and the respective owners.

Suppose a vCenter Server administrator wants to increase the stats level in vCenter Server for some interactive debugging. The vCenter Server administrator merely posts a message to the vCenter group, notifying all relevant stakeholders that a change event is going to occur. With SCX grabbing log data from Log Insight and parsing it for warning messages, if "slow DB statements" occur, the vCenter Server administrator is notified. Moreover, if the database administrator suddenly sees unusually high disk activity, she is able to correlate it with the posted message from the vCenter Server administrator regarding the change in stats level. It is important to note that at present, the act of changing the statistics level does not create a task event in vCenter server, so it is difficult to use automated scripts and the current API to detect that the statistics level has been changed.

4.4 Propagating Events to Stakeholders

When an ESX host enters maintenance mode, its VMs are automatically migrated to other hosts ("VMotion"), and separate tasks are generated for each VMotion. A VM owner who suddenly experiences slowness might be the victim of such a collateral VMotion but might not know that this is the reason her VM has had a temporary slowdown. Because SCX monitors both inventory changes and general task events, it automatically updates the connections within Socialcast. SCX also posts a message to the host and VM pages regarding the host entering maintenance

mode and the VMotions that result. Thus, even though the original action (enter maintenance mode) is performed on a host, any VM stakeholders who are affected get notified that a VMotion has occurred due to maintenance mode. If no slowness is encountered, the migration messages serve as preemptive warnings about possible performance issues rather than signaling a performance problem. A screenshot of this scenario is shown in Figure 4.

Figure 4. Propagating events to stakeholders. A VM owner is not subscribed to updates from the host running the VM, but if the VM is migrated because maintenance mode was entered, the VM owner is notified.

4.5 Integration with Bugzilla

Because SCX stores connections among users and entities in Socialcast, it can use Socialcast as a lookup service to identify stakeholders when they file a trouble ticket. For example, suppose a host owner wants to add a hard drive to an existing host and wants to notify the vCenter Server administrator and other administrators that might be affected by the host going down. Figure 5 shows a simple example of modifying bug-tracking software (in this case, Bugzilla [25] to use such a lookup service. We add a Stakeholders field to Bugzilla. When we enter an entity's username in the appropriate field, Bugzilla can issue a request to Socialcast to retrieve all human followers of this VM. After the human followers are retrieved, their email addresses are retrieved and prefilled in the Stakeholder box of Bugzilla. Because all of the data is already within Socialcast, it would also be straightforward to import the data directly into Bugzilla.

Figure 5. Bugzilla Integration and Socialcast as a lookup service. When a user wishes to file a bug related to a given asset (here, the vcenter SSO VM with the Socialcast username vcenterssoservervm), the stakeholders field is automatically populated with the email addresses of the human users who are following this VM.

5. Related Work

The concept of adding virtualization entities to a social network and connecting humans to their machines is first mentioned in a paper by Soundararajan and Muppalla [12]. Follow-on work by Soundararajan and Spracklen [13] takes the simple set of connections in a social network (membership and following) and explores the use of the Neo4j [5] graph database as a back end for storing these connections. They observe that many common IT questions can framed in terms of graph traversals and are therefore a good match for a graph database. We build upon this work by adding new sources of information (e.g., vRealize Operations Manager, vCloud Director, and Log Insight) as entities, and we build upon the social media metaphor by utilizing these entities and connections for collaborative debugging.

Socialcast already has an API that enables integration with business systems [10] [11], similar in theme to our Bugzilla integration. Our work is one step toward a similar such integration with vSphere. Other possible integrations include a collaborative environment similar to a Google Hangouts [2], a so-called Social War Room (a term coined by colleagues [4] [7]). The term is reminiscent of an IT war room, in which relevant parties are gathered in response to an IT escalation. In a Social War Room, in response to an escalation, the connections in the social network are used to determine which parties to notify (e.g., the network administrator responsible for a given switch is appropriately notified). A group is then created on the fly in response to the escalation, with the relevant parties able to view domain-specific data through the Socialcast portal (e.g., a storage administrator sees storage metrics, while a network administrator sees network statistics). Such a group exists for the duration of the issue, just as in an actual IT war room. Our work differs from a Social War Room in one fundamental way: A Social War Room is typically driven by a particular event and does not serve as the audit trail or the notification mechanism for day-to-day monitoring. In contrast, we organize all information according to assets (such as VMs and hosts), and we let users subscribe to assets that they care about. In theory, any actions that occur to an asset can result in notifications to relevant stakeholders, and any asset can be examined to determine who those stakeholders are. Stakeholders often listen to only a subset of an entire inventory or set of products, thereby limiting the amount of information that a single user must absorb. For an administrator who must listen to a large number of entities, we must develop better filtering capabilities, and we can leverage existing work in assessing relevance in social networks [23] for this purpose.

vSphere Web Client can be seen as a single pane of glass over an infrastructure, providing similar monitoring capabilities to what we propose using Socialcast. There have been proposals to embed social media into the existing vSphere Web Client to allow collaboration (suggested by colleagues [3] [6]). Although this approach is similar to our work, using the web client for collaboration has one main drawback: The web client is typically intended for use by administrators, not by end users, so connecting a user

with her VM would perhaps require the user to have administrative access, which is typically less common than giving a user an account on an in-house social media portal. One other potential drawback is that the web client is a full-blown management portal and does not have the same ease of use as a simple social portal. While the latter point is admittedly subjective, customers have indicated informally that a social web portal is more intuitive for this class of initial triage and debugging [1].

The use of Socialcast as a lookup service might appear to be redundant when an administrator is using Active Directory (AD), because groups and assets can also be maintained within AD. However, the advantage of Socialcast is that end users can add and remove themselves on the fly to groups without requiring administrative approval.

6. Conclusions and Future Work

Administrators are constantly frustrated by a simple question: When an infrastructure emergency happens, who needs to be contacted and how? In this paper, we address this issue by looking at virtual infrastructure as an "Internet of things" in which all actors (human, machine, or business system) have an identity and are connected to one another. We observe that although a variety of tools exist to manage and monitor an environment, these tools do not form a cohesive monitoring framework, and they also ignore collaboration as a debugging aid. We propose a solution that leverages social media to achieve this goal. Our SCX prototype takes monitoring information from vRealize Operations Manager and Log Insight, couples this data with inventory information from vCenter Server, and then posts this data to Socialcast. Administrators can collaborate on the issue and keep an audit trail of operations within Socialcast. Moreover, SCX parses messages before sending them to Socialcast so that it can provide rich links to entities (e.g., a link to the vSphere Web Client page for a given VM). Rather than simply using Socialcast as a "war room," we propose that it be used to provide a simple "one-stop shop" for viewing the life cycle of a VM or host, including user-initiated events and error events. This paper is not just about using Socialcast as a UI, or creating glue logic that can pull data from disparate sources and push them onto a single pane of glass. Rather, it is about creating a truly collaborative work environment that incorporates available data sources to provide a suite-like experience.

Although we have only considered small use cases involving vRealize Operations Manager, Log Insight, vCenter Server, and vCloud Director, our next step is to integrate additional products into SCX. Specifically, we are interested in adding automation engines such as VMware vRealize™ Automation™ [15]. We are targeting these sorts of applications because they enable a user to create a workflow that includes approvals (e.g., before a VM can be reconfigured, the IT administrators responsible for the VM should be notified). By inserting a step in which a Socialcast notification can be sent before a VM is actually reconfigured, we avoid issues in which an administrator reconfigures a VM and posts a message after the fact, rather than obtaining permission from appropriate stakeholders beforehand. As the amount of data grows and the connectivity becomes more complex, we can consider the using of a graph database to supplement the MySQL database of Socialcast.

A somewhat longer-term project expands the scope of the "Internet of things" to external data sources such as news feeds. For example, a weather-related RSS feed can be sent to SCX, which can then parse such information and query Socialcast to determine if any data centers are in the path of a storm. Because a severe storm can cause power outages and therefore service outages, it is important know about such storms and take preemptive action (e.g. , moving VMs to other data centers). For military applications, one could envision determining all of the data centers that would be affected if a conflict were to break out. Although it might seem a bit futuristic, this example is inspired by actual discussions with VMware customers.

References

1. Customer feedback, VMware Customer Advisory Council, June 2012.
2. Google Hangouts. https://www.google.com/hangouts/
3. Halachliyski, Deyan. VMware, personal communication, October 2013.
4. Masnik, Marc, VMware, personal communication, December 2012.
5. Neo4j. http://www.neo4j.org/
6. Papadmi, Johnny, VMware, personal communication, October 2013.
7. Paynter, Andrew, VMware, personal communication. January 2014.
8. Redis key-value store. http://redis.io
9. Rodriguez, Marko. Supporting the Emerging Graph Landscape. http://markorodriguez.com/
10. Making SharePoint Social: Integrating Socialcast and SharePoint Using Reach and API. http://blog.socialcast.com/making-sharepoint-social-integrating-socialcast-and-sharepoint-using-reach-and-api
11. Socialcast Developer API. http://www.socialcast.com/resources/api.html
12. Soundararajan, R. et al. A Social Media Approach to Virtualization Management. VMware Technical Journal, November 2012.

13. Soundararajan, R. and Spracklen, L. Simplifying Virtualization Management Using Graph Databases. VMware Technical Journal, June 2013.

14. Soundararajan, R. and Spracklen, L. Revisiting the Management Control Plane in Virtualized Cloud Computing Infrastructure. In Proceedings of IISWC 2013, September 22–24, 2013. Portland, OR. USA.

15. VMware vRealize Automation. http://www.vmware.com/products/vrealize-automation

16. VMware vCloud Director. http://www.vmware.com/products/vcloud-director

17. VMware vRealize Operations Manager. http://www.vmware.com/products/vrealize-operations

18. VMware vRealize Log Insight. http://www.vmware.com/products/vrealize-log-insight

19. VMware Socialcast. http://www.vmware.com/products/socialcast

20. VMware vSphere. http://www.vmware.com/products/vsphere

21. vSphere API Reference Documentation. https://www.vmware.com/support/developer/vc-sdk/visdk41pubs/ApiReference/index.html

22. VMware vSphere Product Documentation. http://www.vmware.com/pdf/vsphere5/r55/vsphere-55-configuration-maximums.pdf

23. Wang et al. Learning Relevance from a Heterogeneous Social Network and Its Application in Online Targeting. In Proceedings of SIGIR. July 24–28, 2011, Beijing, China.

24. Wikipedia. Internet of Things. http://en.wikipedia.org/wiki/Internet_of_Things

25. Bugzilla http://www.bugzilla.org/

Directions in Mobile Enterprise Connectivity

Craig Newell
VMware Inc.
craign@vmware.com

Abstract

One, if not the most important, attribute required for the enterprise use of mobile devices is access to network-accessible resources offered within the enterprise. Physical connectivity has been made widely accessible, with convenient and affordable Internet access available to mobile devices via technologies such as UMTS/LTE and WiFi. The predominant technology to enable this access to enterprise resources is the virtual private network (VPN) [17]. This paper describes some of the recent mobile-device VPN architectures and presents a proposal for future evolution: the "per-app" VPN with microsegmentation.

1. Motivations for VPN

To understand the currently available mobile VPN architectures, the driving factors behind the development should be examined. These include, in no particular order:

- Reachability
- Persistence
- Security
- Usability

1.1 Reachability

The architecture of the Internet is conceptually very simple, with every node directly addressable and accessible from all other nodes. This can be represented in the mobile device accessing enterprise services directly, as in Figure 1.

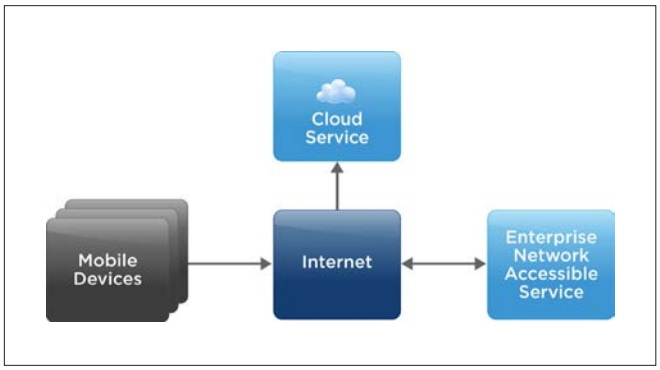


Figure 1. Simplistic Network Model

Due to the shortage of IPv4 address space and the easy availability of private IPv4 addresses [9], Network Address Translation (NAT [13]) has been extensively deployed, removing the directly addressable characteristic of the original Internet architecture. This has resulted in a logical network for mobile devices, as in Figure 2.

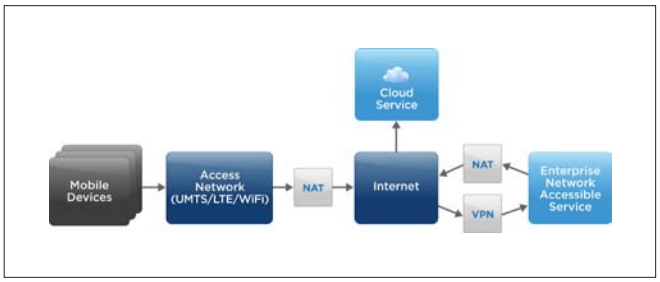


Figure 2. Actual Network Model

NAT generally only supports connections initiated from the "private" side to the "public" side using a subset of IP protocols (normally just a subset of ICMP, UDP, and TCP). This greatly limits the use of other IP protocols for VPN purposes, often forcing traffic to be tunneled via HTTP/HTTPS, because this is almost always transported for web browser use.

A VPN resolves both the reachability and protocol restrictions of NAT by connecting mobile devices logically to the inside of the "private" network.

1.2 Persistence

Most mobile devices can connect to the Internet via one or more wireless technologies (e.g., UMTS/LTE and WiFi) or wireless and wired in the case of laptop computers (see Figure 3). The exact configuration of these network-access methods dynamically changes with network conditions.

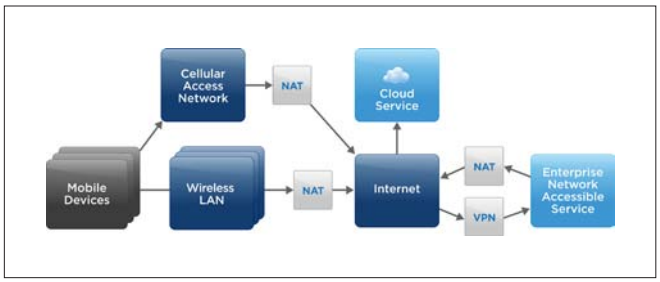


Figure 3. Commonly Available Access Network

The changing network connectivity from the mobile device results in a lack of connection persistence for applications, because the IP addressing differs for each connectivity method. This has been attempted to be addressed at Layer 3 [14] by standards such as Mobile IP [11] or at Layer 4/5 with efforts such as Multipath TCP [12]. These have not been deployed widely and/or end up with mobile applications having to manage the changing connectivity within the application itself.

The tunneling inherent in VPNs can provide persistence for applications, assuming suitable VPN session management has been used—significantly simplifying the required networking logic within applications.

1.3 Security

Information security is often defined as protecting three properties [8]:

- Confidentiality
- Integrity
- Availability

This information security with respect to mobile devices focuses on the confidentiality and integrity properties. The availability aspect is normally ignored because the most common availability issue is the battery going flat, which requires physical device usage protocols to address (e.g., always charging the device every night).

The confidentiality and integrity properties are normally addressed by adding a cryptographic protocol such as TLS [16] under the application protocol. However, practicalities such as enterprise security (e.g., data-loss prevention), availability, and scaling (e.g., TLS offload and load balancing) break the end-to-end security model suggested by the use of TLS (see Figure 4).

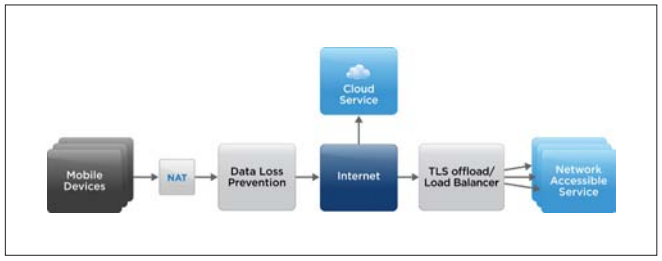


Figure 4. Actual End-to-End TLS Network

This is forcing applications to adopt an additional cryptographic protocol for use within TLS to achieve end-to-end confidentiality and integrity. The use of a VPN that provides an "outer layer" of security can provide hardening to applications using TLS that reduces some of the need for a separate inner encryption layer.

This extra encryption layer added by applications is removing the effectiveness of examining traffic and hence the effectiveness of traffic monitoring for security purposes. If the content of the traffic cannot be monitored, then the ability to segregate network traffic from individual applications to isolated networks is becoming the only practical solution (see Section 2.3).

1.4 Usability

Mobile devices are used for enterprise use for any number of reasons. Sometimes they are required to perform a certain task (e.g., retail stocktaking on a ruggedized portable terminal), but often the major uses are just personal productivity (e.g., checking email and calendar on a smartphone). When used for personal productivity, usability becomes really important. In particular:

- Seamless operation
- Authentication

1.4.1 Seamless Operation

The users of mobile devices have little interest in the complexities of networking beyond the simple monitoring of wireless signal strength. The consumer-driven expectation is that applications just "work" without any additional interdependencies such as launching and tracking the state of a VPN client—necessary so that applications can correctly access their network services.

Enterprise connectivity can be added within applications via software development kits (SDKs), but this is not scalable for third-party applications to maintain variants for each VPN vendor. This is pushing the model of independent VPN clients with "on-demand" initiation to make them transparent to the end user and the application.

1.4.2 Authentication

No one wants to enter a complex password and a two-factor authentication token just to view the next scheduled calendar event on a mobile device. When VPN has been deployed for mobile workers on laptops, cumbersome authentication methods such as external second-factor tokens have been commonly required. However, increasing acceptance by the security community that a certificate stored in a hardware-protected location on a mobile device that has a device lock by PIN code (or fingerprint) is a good balance between security and usability when coupled with simple access policies based on geo-location and time.

Enhanced security can be layered—without impacting the common user experience—via adaptive step-up authentication when simple time and geo-location policies are not flexible enough. (Executives accessing email from the other side of the world in the middle of the night does happen, even if rarely.)

2. Mobile VPN Architectures

VPNs are used in many different scenarios. In this section we review some of the possible architectures used for mobile-device access.

2.1 Client Platform Support

Mobile devices have constrained and restricted operating systems compared to desktop and server environments, to improve robustness and usability. In this section the VPN implementation models of the two most common mobile-device platforms, Apple iOS and Google Android, are detailed.

2.1.1 Apple iOS

Limited VPN support has been available on iOS since iOS 3, when Apple added a built-in VPN client. iOS 5 extended this support to third-party VPN clients with the addition of a VPN plug-in API. Two modes of operation are provided as of iOS 7:

- **Full-device** – In full-device mode, the VPN client can capture and inject Layer 3 frames (IP packets) from the iOS kernel and in doing so is able to tunnel all the traffic from all applications running on the device (see Figure 5) without any modification or configuration of the applications.

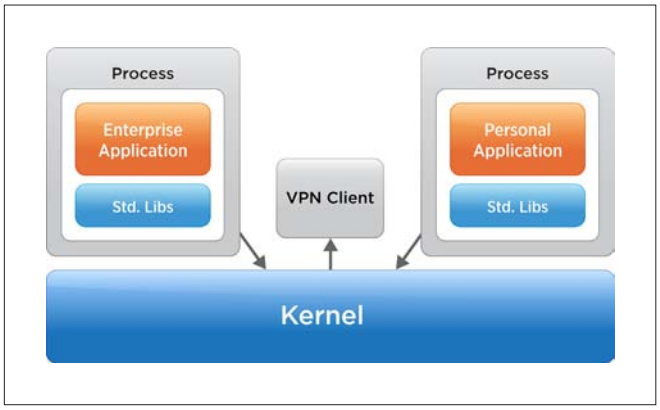


Figure 5. Full-Device VPN

- **Per-app** – iOS also provides a Layer 5 "per-app" VPN model. In this mode, individual applications "managed" via mobile device management (MDM) can have their networking traffic redirected to a third-party VPN client. This redirection is implemented within the standard iOS libraries (`CoreFoundation`) such that applications themselves do not need to be modified and without direct kernel involvement (see Figure 6).

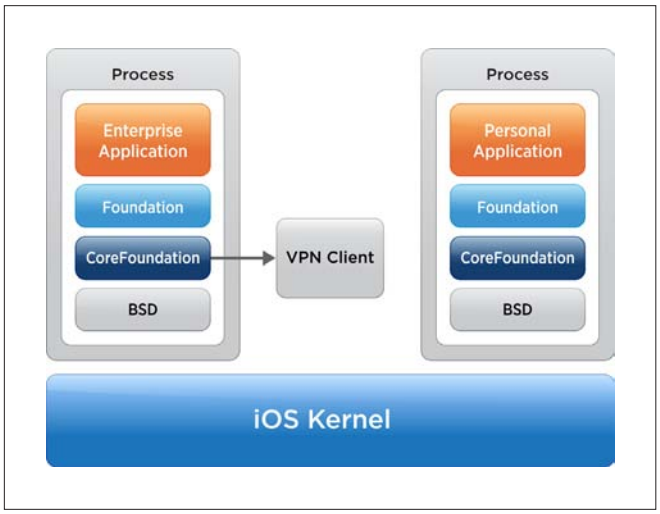


Figure 6. iOS Per-App Device VPN

An alternative implementation of "per-app" VPN on iOS is at the application level using SDKs such as those offered by AirWatch, F5, and NetScaler. However, this requires that applications be explicitly modified during development to support connectivity via the SDK and causes distribution issues with third-party applications. In addition, SDKs are becoming less compatible over time as iOS sandboxes libraries with a large security surface into separate processes (e.g. QuickLook, `WKWebView`).

2.1.2 Google Android

The Android platform has contained a standardized third-party VPN client model since Ice Cream Sandwich [2], with some limited enhancements available as of Android Lollypop [1]:

- **Full-device** – Limited VPN client support has been available on Android since Gingerbread on some devices, but it was not until Ice Cream Sandwich that a standardized API [3] was made available for third-party VPN clients. Since then, "full-device" VPN has been supported just as on iOS (see Figure 5) but with an Android user experience for control and status.
- **"Workspace"** – In Android Lollypop, the VpnService API was extended to support the whitelisting and blacklisting of applications accessing the VPN. With this support, network access via the VPN client can be restricted to a list of applications (a "workspace"). The VPN client still operates at Layer 3 (IP frames), unlike iOS's "per-app" VPN, and the VPN client is unable to distinguish traffic from individual applications (see Figure 7).

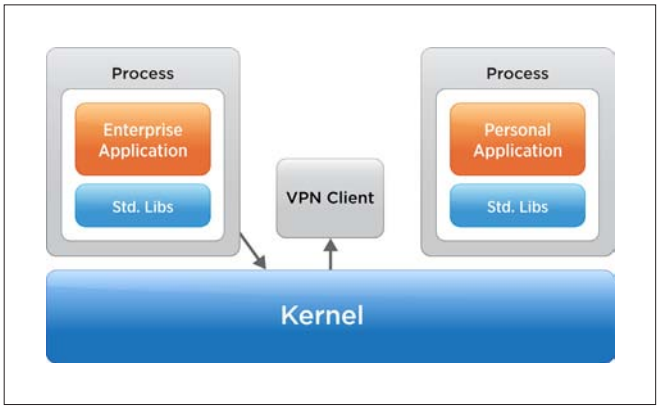


Figure 7. "Workspace" VPN

- **Per-app** – The current Android VPN model does not directly support a true "per-app" model whereby traffic can be identified and controlled on a per-individual-application basis, but it does have enough support to allow this to be reverse-engineered. This can be done by taking a "user space NAT" implementation used to provide "share with host" networking for virtual machines on PCs and adding a filtering layer based on the originating process and finally a Layer 5 VPN client (see Figure 8).

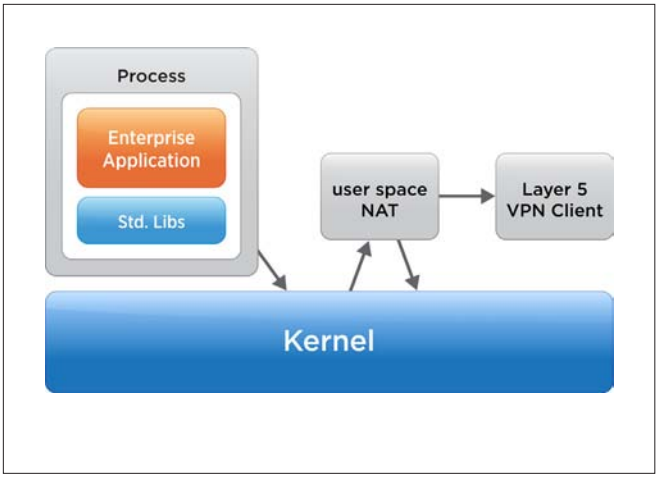


Figure 8. Android "Per-App" Device VPN

2.2 VPN Gateway Architectures

The concept of a VPN gateway is well understood, with the tunneling protocols operating at many different OSI layers (see Figure 9).

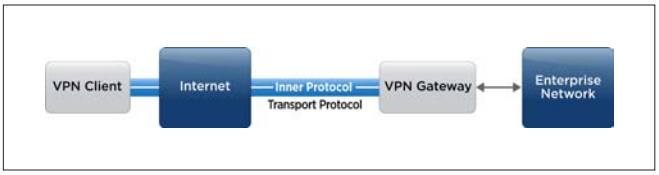


Figure 9. Abstract VPN Gateway Model

In recent times, with the widespread adoption of NAT, the transport protocol for VPN has been restricted in practice to UDP and/or TCP. It is also common to find simplistic firewalls that restrict options further to just TLS on TCP port 443 where UDP, such as encapsulated IPsec or DTLS, is not supported. However, the traffic transported within the inner protocol does not have these limitations and can still be of many different protocols often grouped according to the OSI level of the traffic that is being tunneled.

2.2.1 Layer 3 Gateway

A "classic" VPN tunnels traffic at OSI Layer 3, which is normally IP frames. This is the "full-device" VPN model supported by both iOS and Android. The IP configuration is normally an address from the enterprise network providing support for connections both from and to the mobile device. However, in the case that inbound connections are not required, the deployment configuration can be greatly simplified by integrating a NAT implementation and another private IPv4 address space for the clients.

2.2.2 Layer 5 Gateway

The iOS "per-app" (see Section 2.1.1) VPN model operates at OSI Layer 5, just like the common SOCKS proxy protocol [15]. Adding TLS and authentication to SOCKS, this can be used for VPN without requiring the complexity of configuration and deployment that a Layer 3 solution requires when used without NAT.

2.2.3 Layer 7 Proxy/Gateway

The dominant use of HTTP/HTTPS by browsers and applications has driven the use of the HTTP Proxy protocol as a "VPN." This can be used in a forward proxy configuration (using GET, POST, etc.) for HTTP traffic or in a "tunneling" mode with the CONNECT command. This is especially popular when the VPN client is a SDK that is used by the application.

2.3 Segmented Gateway

In the normal model of a VPN gateway (see Figure 9), all traffic from the VPN gateway is forwarded to a single "private" network. In practice, this is not always the case, and it is normally coarsely segmented into access groups (e.g., different user classes such as contractor and employee). However, as was discussed in Section 1.3, there is great value in segregating traffic according to the originating application. Because there are many different applications that perform the same basic task (e.g., iPhone, iPad, Android), grouping them into "service networks" can help with overall manageability (see Table 1).

SERVICE NETWORK	VLAN #	DESCRIPTION
Device Compromised	1	Captive portal for out-of-policy devices (e.g., jailbroken)
Internet Only	2	No intranet access, only filtered Internet access
Intranet Only	3	No internet access, only internal sites
Finance	4	Application access to Oracle iExpense, etc.
R&D	5	Application access to JIRA ticket tracking, source code
Sales & Support	6	Application access to Internal documentation and knowledge base

Table 1. Example Network Segmentation

In this case, the model of the VPN gateway ends up looking like Figure 10, with the VPN gateway having interfaces on multiple internal segments (VLANS).

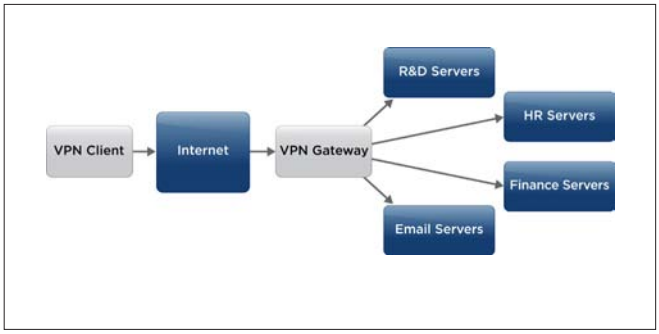


Figure 10. Segmented VPN

3. Summary

Mobile computing is replacing the desktop as the primary computing device for many [7], and the adoption of a mobile-friendly VPN has the potential to make enterprise use as convenient as the desktop. This use of VPN will result in

- Increased usage of mobile devices within the enterprise
- Improved user experience for enterprise employees
- Improved security from segmentation and reduced use of consumer applications for enterprise use
- Reduction of complexity for mobile-application developers, resulting in more enterprise applications

This paper has highlighted the key attributes of VPN for mobile-device use, including: clients with on-demand activation, certificate-based authentication, step-up authentication, and "per-app" segmentation; and gateways with NAT and multiple-network-segment support.

4. Future Work

This review of the current state of VPN architecture for mobile devices has revealed a large opportunity for the optimization of VPN in real-life deployment scenarios.

The inner VPN protocol is often very simple and can be enhanced in many areas, such as low-latency stream flow control, traffic prioritization, and reliability emulation (e.g., dropping UDP frames when traffic is congested over TCP transport). The transport protocol also needs to be examined, because mobile networks are both not-lossy (e.g., UMTS/LTE with reliable delivery) and lossy (e.g., WiFi), suggesting that some level of forward error correction [10] should be added to improve performance (e.g., [4] [5] [6]).

On the gateway side, closer integration with software-defined networking (SDN) in increasing the segmentation granularity (microsegmentation) and with the presentation of user, application, physical location, and so on attributes to the SDN policy layer should improve manageability and security.

Acknowledgments

Although VPN has been used on mobile devices for a long time, it is great to see Apple in iOS pushing VPN with a seamless end-user experience, making it practical for non-technical users. I would like to thank AirWatch for giving me the opportunity to shape a mobile VPN strategy; Tom Corn of VMware® NSX™ security for driving home the future importance of network segmentation; and VMware EUC CTO Kit Colbert for pushing for networking and mobility to work more closely within VMware.

References

1. Google. Android Lollipop – Android Developers, 2014. [Online; accessed 17-Nov-2014].
2. Google. Ice Cream Sandwich – Android Developers, 2014. [Online; accessed 17-Nov-2014].
3. Google. VpnService – Android Developers, 2014. [Online; accessed 17-Nov-2014].
4. Minji Kim, Muriel Médard, and João Barros. Modeling network coded TCP throughput: a simple model and its validation. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, pages 131–140, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
5. Dzmitry Kliazovich, Magda Bendazzoli, and Fabrizio Granelli. TCP-aware forward error correction for wireless networks. In *Mobile Lightweight Wireless Systems*, pages 68–77. Springer, 2010.
6. Benyuan Liu, D.L. Goeckel, and D. Towsley. TCP-cognizant adaptive forward error correction in wireless networks. In *Global Telecommunications Conference, 2002. GLOBECOM '02*. IEEE, volume 3, pages 2128–2132 vol.3, Nov 2002.
7. Steven Norton. A Post-PC CEO: No Desk, No Desktop, 2014. [Online, accessed 11-Dec-2014].
8. Chad Perrin. The CIA Triad, 2014. [Online; accessed 24-Nov-2014].
9. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. Updated by RFC 6761.
10. Wikipedia. Forward error correction – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 11-Nov-2014].
11. Wikipedia. Mobile IP – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 13-Nov-2014].
12. Wikipedia. Multipath TCP – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-Nov-2014].
13. Wikipedia. Network address translation – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-Nov-2014].
14. Wikipedia. OSI model – Wikipedia, the free encyclopedia, 2014. [Online; accessed 11-Nov-2014].
15. Wikipedia. SOCKS – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 17-Nov-2014].
16. Wikipedia. Transport Layer Security – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 19-Nov-2014].
17. Wikipedia. Virtual private network – Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 11-Nov-2014].

Notes

Notes

Notes


vmware®

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

http://labs.vmware.com/academic