2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

A High-Performance Media Streaming Architecture based on KVM

Woo-Yeong Jeong, Youngjae Lee, and Jin-Soo Kim
College of Information and Communication Engineering
Sungkyunkwan University
Suwon, South Korea
Email: {wooyeong, yjlee}@csl.skku.edu, jinsookim@skku.edu

***Abstract*—A media streaming server can be implemented on a virtual machine for the ease of resource management. However, simply running a media streaming server on a virtual machine has two problems; the duplicate data in file caches of virtual machines and the performance degradation caused by the virtualization overhead. In order to resolve these problems, this paper proposes a high-performance media streaming architecture based on KVM. First, we implement a shared cache among virtual machines in order to eliminate the duplicate cached data. Second, the `sendfile()` operation is offloaded to the hypervisor to reduce the virtualization overhead in I/O operations. Our evaluations with D-DASH datasets show that the performance of a media streaming server in the proposed architecture is increased by up to 30% as compared to that of the conventional media streaming server that simply runs on a virtual machine.**

***Keywords*—Virtualization, Media streaming, TCP socket offloading, KVM**

I. Introduction

Media streaming is one of the most popular services in the Internet. A number of media streaming service providers such as *YouTube, Netflix*, and *Hulu* have been successful over the last decade. With the emergence of media streaming services, the traffic of media streaming accounts for a significant portion of the total Internet traffic and is expected to grow rapidly. In particular, Cisco presents that the media traffic is 66% of all consumer Internet traffic in 2013 and will be 79% in 2018 [1].

In order to achieve high quality of service (QoS) in the media streaming service, it is essential to manage the resources dynamically according to the real-time amount of streaming requests. To ease the dynamic resource management, media streaming servers are usually implemented on a cloud computing platform based on a virtualization technology, where the servers run on VMs (virtual machines), not directly on physical machines. Through existing virtualization techniques, computing power such as CPU cores allocated to each server can be dynamically managed and additional VMs can be deployed or terminated in real time. When media streaming servers are operated directly on physical machines, it is practically impossible to do so.

However, the approach that simply employs a virtualization technology has the following two problems. Typically, media contents are stored in file storage servers that are remotely located. Due to performance issues, a media streaming server caches media files retrieved from file storage servers in its local file cache. When a number of servers handle the requests for the same media contents, the same files are cached in the local file cache of each server. If the VMs of the servers reside in the same physical machine, actually there is the duplicate cached data in the local disks of the physical machine. If the duplicate is eliminated, the overall streaming performance can be much improved as more files can be cached.

Another problem is performance degradation due to the virtualization overhead. In a virtualized environment, the operating system of a VM runs in an unprivileged mode so that it cannot access hardware devices directly. All hardware accesses are conducted by *hypervisor* running in a privileged mode. Whenever a media streaming server reads/writes data, the hypervisor performs the actual I/O operations. Such intervention of the hypervisor causes a notable overhead.

In order to resolve these problems, this paper proposes a high-performance media streaming architecture based on KVM. First, we design a shared file cache among VMs by the use of *VirtFS* [2]. VirtFS enables VMs to share file systems of the host system. Also, to address the performance degradation caused by the virtualization overhead in I/O operations, we offload `sendfile()` system call to the hypervisor. During a `sendfile()` system call, lots of I/O operations are generated so that the intervention of the hypervisor occurs frequently. In the proposed architecture, `sendfile()` operations are performed by the hypervisor. As a result, the number of hypervisor's interventions is effectively reduced so that the performance degradation is also diminished.

II. Background

A. Media Streaming

The most popular standard technology for media streaming is Dynamic Adaptive Streaming over HTTP (DASH). DASH is a technology for an adaptive bit rate streaming, which enables high quality of media contents over the Internet [3]. DASH detects the network bandwidth and computing power of a client in real time and adjusts the quality of media contents sent to the client accordingly. In DASH, a media file is divided into one or more segments. Each segment contains a certain interval of playback time of the media file's contents. For each interval, segments are encoded at multiple bit rates. A client can access the segment corresponding to any playback position at the highest bit rate possible that can be smoothly downloaded.

1066-6192/15 $31.00 © 2015 IEEE
DOI 10.1109/PDP.2015.90

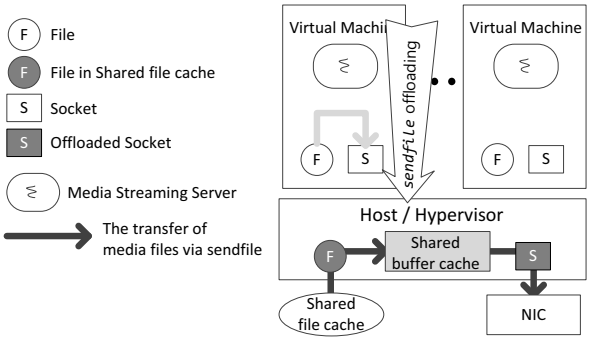

Fig. 1: Proposed media streaming architecture

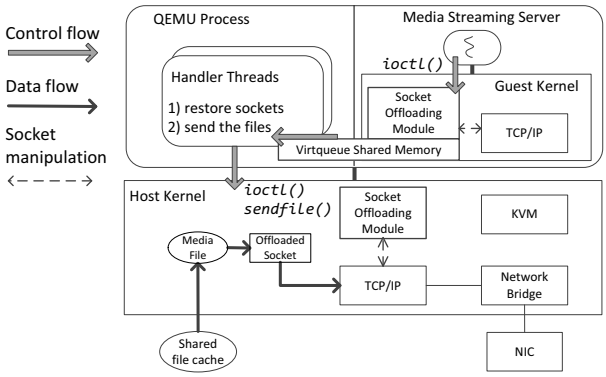

Fig. 2: The offloading of `sendfile()` operations

B. Kernel-based Virtual Machine (KVM)

Kernel-based Virtual Machine (KVM) is a representative virtualization solution for Linux [4]. For high I/O performance, KVM supports paravirtualized devices, based on *Virtio* [5]. One of the major roles of Virtio is to provide shared memory, called *virtqueue*, between the hypervisor and a VM. A device driver in a VM transfers I/O requests to the hypervisor via its virtqueue, and the hypervisor handles the requests and returns their results to the driver via the same virtqueue.

VirtFS is a paravirtualized filesystem based on Virtio [2]. VirtFS enables VMs to share file systems of the host system. A VM can mount a certain directory of the host's file system to its own directory so that it can access part of the host's file system. The file system driver of VirtFS in a VM transfers file-level operations to the hypervisor via its virtqueue.

III. Media Streaming Architecture based on KVM

A. Overall Architecture

This paper proposes a high-performance media streaming architecture, depicted in Fig. 1, in order to address the aforementioned problems. In the proposed architecture, there is a shared file cache in the host system, which all VMs utilize. All media files are cached not in the file cache of each VM, but in the shared file cache.

Also, I/O operations for media streaming are offloaded to the hypervisor to reduce the hypervisor intervention. Given a client's streaming request, the media streaming server notifies the hypervisor of the information of the requested media files. Then, the hypervisor retrieves the media files from storage and transfers them to the client. After finishing to send the media files, the hypervisor notifies the server that the transfer is completed. In the proposed architecture, the number of hypervisor interventions is effectively reduced so that the overall streaming performance is improved.

We implemented the prototype of the proposed architecture based on KVM. In the prototype, a media streaming server transfers the requested media files to a client via the `sendfile()` system call. The following subsections describe the detailed implementation of the prototype.

B. Shared Cache among Virtual Machines

We implemented a shared file cache among VMs by the use of VirtFS. In the prototype, VMs share a file system of the host system via VirtFS, where media files are cached. When a media streaming server in a VM reads a certain media content from file storage servers, the media files are cached in the shared file system. After that, if servers in other VMs issue read operations on the media files, the read operations are handled by the cached data in the shared file system. Therefore, in the prototype, the shared file system acts practically as a shared file cache among VMs. Also, the shared file system is accessed through the buffer cache of the host system so that the buffer cache practically works as a shared buffer cache.

C. The Offloading of Sendfile Operations

In the prototype, `sendfile()` operations in media streaming servers are offloaded to the hypervisor of the host system in a similar way to how Gamage et al. implement them on Xen virtualization environment [6]. Given a client's streaming request, the server informs the hypervisor (i.e., QEMU process in Fig. 2) of the information of the requested media file through virtqueue and hands over a network connection with the client to the hypervisor. Then, the hypervisor transfers the media file to the client via the `sendfile()` operation and returns the network connection to the media streaming server. Note that we assume media files are cached in the file system of the host system which is shared among VMs by VirtFS so that the hypervisor can access the cached media files without any manipulation. During the offloading of `sendfile()` operations, the most complex process is the handover of the network connection with a client. The details of the handover process are described below.

In order to move a network connection with a client from the media streaming server to the hypervisor, the information of the TCP socket should be delivered to the hypervisor so that the TCP socket can be reconstructed using the information in the hypervisor. We implemented a kernel module, called *Socket Offloading Module*, to extract the information of a TCP socket and to reconstruct the TCP socket with the extracted information.

After the TCP socket is reconstructed in the hypervisor, all the packets destined to the media streaming server should be routed to the reconstructed socket. For the packet redirection, we utilized a network address translation (NAT) technique. Also, when the offloading starts, the original TCP socket

should not receive any packets from the client. The reason is because if the original socket sends an ACK packet, the reconstructed socket in the hypervisor will be invalidated since it has sequence numbers different from those of the client. The TCP socket information is transferred also via the shared memory of virtqueue.

We summarize the overall process of the offloading of the `sendfile()` operation as follows (cf. Fig. 2): 1) A streaming client requests a media streaming server to send a media file. 2) The media streaming server configures a firewall to prevent its original TCP socket from receiving any packets. 3) The server requests the socket offloading module to extract the information of the TCP socket and to convey the extracted information and the information of the requested media file to the QEMU process via virtqueue. The communications with the kernel module are conducted by predefined modes of the `ioctl()` function. 4) The handler thread in the QEMU process asks the module to reconstruct the TCP socket. 5) After the completion of the TCP socket reconstruction, the handler thread configures NAT for the packet redirection. At this point, the TCP connection is successfully handed over from the server to the hypervisor. 6) The handler thread transfers the requested media file to the client via the `sendfile()` system call. 7) After the `sendfile()` operation finishes, the socket is restored to the server in a similar way.

IV. Evaluation

A. Methodology

The evaluations are conducted on a machine equipped with an Intel Xeon Quad-Cores 2.26GHz CPU, 8GB RAM, and a Samsung 470 SSD. We use Ubuntu 13.10 with the Linux kernel 3.12.13 and QEMU 1.7.0. In all evaluations, we assume that media files are already cached in the file cache located in the SSD.

We compare three architectures called BAREMETAL, VM, and PROPOSED ARCH. BAREMETAL represents the architecture where a media streaming server is implemented directly on a physical machine. VM denotes the case where a media streaming server is simply run on a VM. PROPOSED ARCH represents our proposed architecture as depicted in Fig. 1.

B. Micro-benchmark

We implemented a micro-benchmark to evaluate the performance of file transfers in each architecture. The in-house micro-benchmark measures the bandwidth while downloading various sizes of files from the media streaming server of each architecture. Fig. 3 depicts the bandwidth when the file size is varied from 4KB to 200MB.

When the file size is smaller than 64KB, the bandwidth of the proposed architecture is lower than that of the VM architecture. The reason is that the actual time for the file transfer is too short so that the overhead of the offloading of `sendfile()` operations is dominant in the overall performance. The time overhead for offloading a `sendfile()` operation is about 1.9ms, which is required to hand over the network connection with a client to the hypervisor of the host system and vice versa. When the file size is 4KB and 64KB, the time overhead is 71% and 48% of the total elapsed time, respectively.

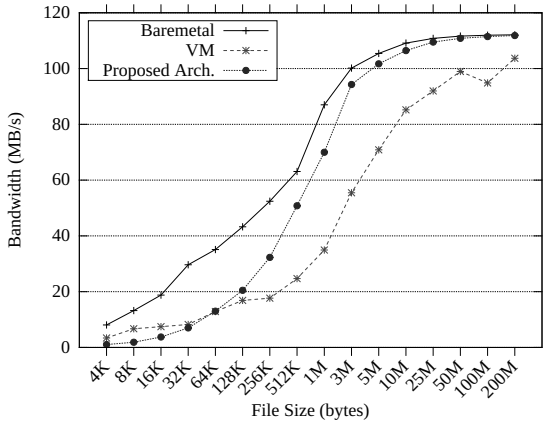

Fig. 3: File transfer bandwidth

Dataset	Smart TV	Tablet	Mobile Phone
Segment length	15 sec	10 sec	6 sec
Video quality	6.0 Mbps	1.2 Mbps	400 kbps
# of segments	351	526	875
Average size of a seg.	10.5 MB	1.4 MB	288.6 KB
Total size of all seg.	3.6 GB	729.5 MB	246.6 MB

TABLE I: D-DASH dataset descriptions

However, when the file is larger than 64KB, the proposed architecture outperforms the VM architecture. In particular, when the file size is 3MB, the bandwidth is improved by 71% as compared to that of the VM architecture. Moreover, the bandwidth of the proposed architecture is almost the same as the BAREMETAL architecture when the file size is larger than or equal to 25MB. We can confirm that the virtualization overhead is successfully overcome in the proposed architecture.

C. Macro-benchmark using D-DASH Datasets

We evaluated the media streaming performance of each architecture with the Distributed-DASH (D-DASH) dataset [7]. D-DASH provides several dozen sets of segments for various kind of devices such as Smart TV and Mobile phone. All segments of each dataset belong to the same video file. According to the DASH technology, the segments of each set are encoded at the bit rate and resolution which are suitable to its corresponding device.

We selected three segment datasets, *Smart TV, Tablet*, and *Mobile Phone* described in Table I. As the Smart TV usually has a huge screen and a stable network connection, the segments of the Smart TV dataset have the highest bit rate and resolution and the longest length: an average segment size is 10.5 MB and the total size of all segment files is 3.6 GB. The segments of the Tablet and Mobile Phone dataset have lower quality and shorter length. We measured the total elapsed time for a media streaming server to transfer all segments of each dataset to a client by the DASH protocol. Each segment is sent to the client by a single `sendfile()` operation.

Fig. 4 shows the evaluation results. Overall, the performance of the proposed architecture outperforms the VM architecture. Particularly, as the segment size becomes larger,

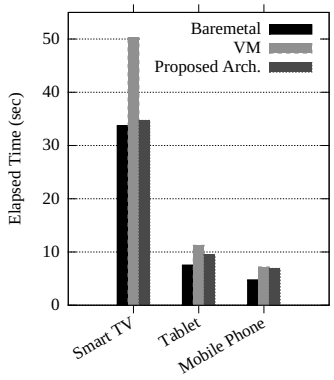

Fig. 4: The elapsed time of each D-DASH dataset

the proposed architecture presents the better performance. In case of the Smart TV dataset, the elapsed time of the proposed architecture is decreased by 30% as compared to that of the VM architecture. Also, the elapsed time of the proposed architecture is longer by only 2.8% as compared to that of the BAREMETAL architecture. In the Mobile phone dataset, the proposed technique is not quite effective because the segment size is too small.

Additionally, in order to show the feasibility of the offloading of `sendfile()` operations, we evaluate the proposed architecture during the live migration of VMs. Note that, in the prototype, if the live migration of the VM which a media streaming server runs on is triggered while the hypervisor conducts offloaded `sendfile()` operations, the migration is performed after the `sendfile()` operations are completed. While downloading the segments of the Smart TV dataset, we triggered the live migration of the VM where a streaming server runs and measured the bandwidth. The live migration took about 13 seconds and the bandwidth is degraded by 28% during the live migration. This is because the network resource is utilized for the live migration as well as the segments transfer. However, we can confirm that the offloading of `sendfile()` operations performs correctly even during the VM live migration.

V. Related Work

There are a few previous work focusing on media streaming servers implemented in a cloud computing platform. Aggarwal et al. utilized a cloud computing platform to operate media streaming servers for an IPTV service [8]. They focused on minimizing the number of servers by the server consolidation technique. Feng et al. suggested a VM migration algorithm for video streaming servers to maximize resource utilization [9]. These papers studied the resource management across multiple physical machines, whereas we focus on the internal architecture of a single physical machine.

There are several researches to mitigate the virtualization overhead caused by the intervention of the hypervisor. Menon et al. optimized the paravirtualized network I/O architecture in Xen by scatter/gather I/O and checksum offloading [10]. In order to minimize the CPU scheduling latency, Kangarlou et al. proposed vSnoop where the hypervisor acknowledges TCP packets on behalf of VMs whenever it is safe to do so [11]. While the aforementioned studies can reduce the overhead, they cannot eliminate the overhead completely. On the other hand, in our proposed architecture, `sendfile()` operations are performed in the hypervisor. Therefore, the overhead can be diminished almost completely. Gamage et al. proposed a simple abstraction for Xen called vPipe to mitigate the overhead in `sendfile()` operations [6]. Similar to our proposed technique, the network connection in VMs is handed over to the hypervisor and I/O operations for file transfers are conducted in the hypervisor. In this paper, the target virtualization platform is KVM. Also, we show more extensive evaluations regarding the performance of the macro-benchmark with D-DASH datasets and the bandwidth of file transfers during the migration of VMs.

VI. Conclusion

This paper proposes a high-performance media streaming architecture based on KVM. We suggest the shared cache among VMs to maximize the cache efficiency and the offloading of `sendfile()` operations to the hypervisor in order to eliminate the virtualization overhead caused by I/O operations. In the proposed architecture, as the virtualization overhead is effectively reduced, the overall performance of a media streaming server is improved by up to 30%, almost reaching the performance of the case where the media streaming server runs directly on a physical machine.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIP) (No. 2013R1A2A1A01016441). This work was also supported by the IT R&D program of MKE/KEIT (No.10041244, SmartTV 2.0 Software Platform).

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2013-2018," Feb. 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[2] V. Jujjuri, E. V. Hensbergen, A. Liguori, and B. Pulavarty, "VirtFS—a virtualization aware file system passthrough," in *OLS*, 2010.

[3] T. Stockhammer, "Dynamic adaptive streaming over http – standards and design principles," in *ACM MMSys*, 2011.

[4] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *OLS*, 2007.

[5] R. Russell, "virtio: Towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 95–103, 2008.

[6] S. Gamage, R. Kompella, and D. Xu, "vPipe: One pipe to connect them all!" in *USENIX HotCloud*, 2013.

[7] S. Lederer, C. Mueller, C. Timmerer, C. Concolato, J. Le Feuvre, and K. Fliegel, "Distributed DASH dataset," in *ACM MMSys*, 2013.

[8] V. Aggarwal, X. Chen, V. Gopalakrishnan, R. Jana, K. Ramakrishnan, and V. A. Vaishampayan, "Exploiting virtualization for delivering cloud-based IPTV services," in *Computer Communications Workshops (INFOCOM WKSHPS)*, 2011.

[9] Y. Feng, B. Li, and B. Li, "Bargaining towards maximized resource utilization in video streaming datacenters," in *IEEE INFOCOM*, 2012.

[10] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in Xen," in *USENIX ATC*, 2006.

[11] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu, "vSnoop: Improving TCP throughput in virtualized environments via acknowledgement offload," in *ACM/IEEE SC*, 2010.