Triple-L: Improving CPS Disk I/O Performance in a Virtualized NAS Environment

Triple-L: Improving CPS Disk I/O Performance in a Virtualized NAS Environment

Dingding Li, Mianxiong Dong, Yong Tang, Laurence T. Yang, Kaoru Ota, and Gansen Zhao

Abstract—Network-attached storage (NAS) provides cyber-physical systems (CPS) with the scalable, efficient and reliable backing storage, such as the mobile virtual desktop based on cloud infrastructure. Within this storage architecture, virtual machine instances (VMs) running in the NAS client usually receive data from the complex physical world and then persist them in the neat cyberspace in NAS sever. In this paper, we propose Triple-L to improve VM disk I/O performance in the NAS architecture. According to the specific storage semantic, Triple-L decouples VM image file into several sub-files at the host layer, and then selectively moves them into NAS client. In such a way, a VM disk I/O request may be proceeded locally in NAS client, instead of walking the external networking path repetitively between NAS server and client. We have implemented Triple-L in a Xen-based NAS system. An accessory solution for dealing with storage failure as well as VM live migration on Triple-L is also discussed and evaluated. The experimental result shows that our work can effectively improve the disk I/O performance of VMs. Meanwhile, it brings moderate overhead for VM live migration.

Index Terms—Cyber-physical systems, virtualization, disk I/O, network-attached storage, image file.

1 Introduction

A typical cyber-physical system (CPS) produces massive and various (big) data from the complex physical world and thus challenges the storage in cyberspace from performance, reliability, real-time and scalability. To meet these requirements, modern CPS usually adopts the network-attached storage (NAS) to act as the backing storage, presenting the feature of cloud-integrated CPS (CCPS) [1] [2]. Taking mobile virtual desktop as an illustrative example, a mobile device in the wild connects a cloud desktop environment via virtual networking computing protocol (VNC), and the end-users could operate their documents in the mobile environment just as in a local-desktop system [3] [4]. Specifically, this kind of desktop environment is often provided by a virtual machine (VM) in the cloud infrastructure, through the maturing technology called virtualization.

Modern public cloud infrastructures usually use virtualization to split the underlying host resource into multiple virtual computing resources [5]. Each of them can be encapsulated as the specific operating system (OS), called virtual machine (VM) or guest operating system, and then be rented to external physical system components (e.g., sensors, mobile device, robots, etc.) for deploying their own cyberspace services or CPS-based applications (e.g., virtual desktop), with hardware-level isolation among each other [6] [7]. In such a server consolidation way, cloud provider in a CPS system achieves improving the hardware utilization, saving the energy cost and enhancing the rental income of computing capabilities in their background systems [8].

At the storage subsystem, as shown in Figure 1, cloud providers in a CPS system incline to storing VM's disk image file in NAS server [9]. Each VM disk image file corresponds to the specific persist data spawned by one or more external physical system components. In this way, the block data from the physical world are focused into a few of dedicated storage servers (namely NAS servers), allowing system administrators to apply a more flexible, reliable and scalable policy to manage the persistent data of a CPS system [10]. For example, compared with direct-attached storage (DAS), in which VM live migration must copy a block of disk data from one source host to another destination host [11], NAS can significantly accelerate this procedure without moving persistent data.

However, due to VM disk I/O request being required to traverse the network link, NAS has the *longer* processing chain on VM's storage path, incurring the extra round-trip latency. On the other hand, NAS also implicitly *narrows* this storage path because a batch of VM disk I/O requests from one NAS client may congest another clients' requests at the central or edge switch, resulting in the jammed and delayed NAS connection. And besides, the transformation of data granularity along a typical virtualized NAS protocol stack will further *complicate* this situation.

Figure 2 illustrates the above procedure. When a CPS-based application running in a VM produces a read/write system call (byte-granularity), denoted by R_{vf}, to its file system, R_{vf} will be converted into a group of virtual *block-level* I/O requests (denoted by $\{R_{vb}\}$) at the block layer (kilobytes-granularity), even though R_{vf} only reads/writes a few of bytes from/to the underlying virtual block device. For example, the target data of R_{vf}, T, crosses the boundary of two adjacent disk blocks in the virtual block device, requiring at least two block-level requests to finish this I/O invocation. After that, frontend driver passes $\{R_{vb}\}$

- *D. Li, Y. Tang and G. Zhao are with the School of Computer Science, South China Normal University, Guangzhou 510613, China. E-mail: {dingdly.scnu, zhaogansen}@gmail.com, ytang4@qq.com.*
- *M. Dong and K. Ota are with the Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 050-8585, Japan. E-mail: {mx.dong, ota}@csse.muroran-it.ac.jp*
- *L. Yang is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada. E-mail: ltyang@stfx.ca*

Manuscript received January XX, 2015; revised XXX XX, 2015.

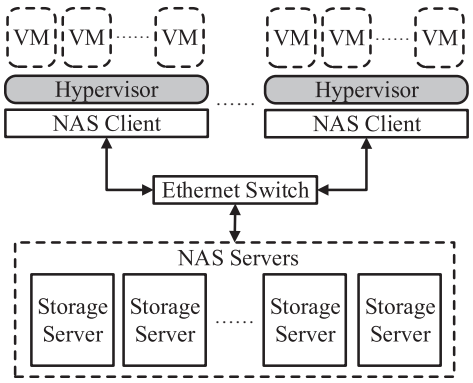

Fig. 1. Following the C/S design, NAS acts as the backing storage for a CCPS system. Generally, in NAS client a VM with CPS-based application will interact with NAS server via networking when the application or its resided file system issues disk I/O request. Thus, the NAS connection may be congested if a large amount of VMs or CPS-based applications issuing the concurrent disk I/O requests.

into the backend driver and then be transformed into a set of physical read/write system calls, denoted by $\{R_{hf}\}$, in hypervisor layer [12]. Finally, backend driver transfers $\{R_{hf}\}$ to the NAS server via the specific NAS protocol (e.g. Samba and NFS). Depending on the particular nature of a network environment (e.g. MTU), $\{R_{hf}\}$ may further being splitted into more network packages, denoted by $\{N_{MTU}\}$. In summary, even the I/O target data of a CPS-based application only covers a few of bytes, the complicated virtualized NAS storage stack will entrain extra innocent data to be transferred over NAS connection [13].

Even worse, in the CPS environment with virtualized NAS system, many external physical system components would connect the shadow VM and transfer the data collected in the wild into cyberspace. During this procedure, physical system varies irregularly alongside the time elapsing, thus producing the sustained and small random I/O in the storage system of cyberspace. Therefore, compared with a general virtualized environment, a CPS environment may stress more scatter and uncertain I/O on the shared storage server. In a general environment, instead, the I/O behavior of VM associates heavily with the end-users operating styles, in which the difference of most of them are veiled by the well-defined application interface, such as the text editor. Finally, a general environment will present the more compact and sequential disk I/O manner than the CPS one.

Therefore, a CPS system with the virtualized backing NAS storage, experiences the *longer* and *narrower* storage path than DAS, especially for those VMs who often carrying *read-modify-write* I/O workloads. A simple write system call issued by one VM's CPS-based application, is required to read the target data from NAS server first and then write them back to server again, incurring multiple round-trip latencies and thus inevitably producing adverse effects on the whole CPS disk I/O performance.

1.1 Case Study

We have studied this issue in a small-scale CCPS which based on the virtual desktop infrastructure (VDI) [14] [15]. A certain number of users (from 1 to 25) boot their VMs within a very narrow time frame. A NAS server and five clients constitute the initial experimental platform. All of them have dual Quad-Core Intel Xeon(R) 1.6GHz processors, 8GB DDR3 RAM, 160GB S-ATA II hard drive with 7200 RPM, and dual Full-duplex Intel Pro/1000 Gbit/s NIC. A Gigabit Ethernet network switch is used to connect them. The driver domain (namely the host OS) in NAS client is running 64-bit CentOS 6.5 distribution and the hypervisor is the newest Xen 4.4 with Linux 3.10.1 kernel. The VMs in NAS client (namely VMs) are running the minimum installation of CentOS 6.5, each with 512MB memory allocation, 16GB disk capacity, QCOW2 image format, Linux 2.6.32 kernel, GNOME3 desktop and the ext4 file system. NFS data transfer buffer sizes are set to 64KB (rsize) and 256KB (wsize). NFS server sets the synchronous flag to `async`, trying to keep the network bandwidth saturated.

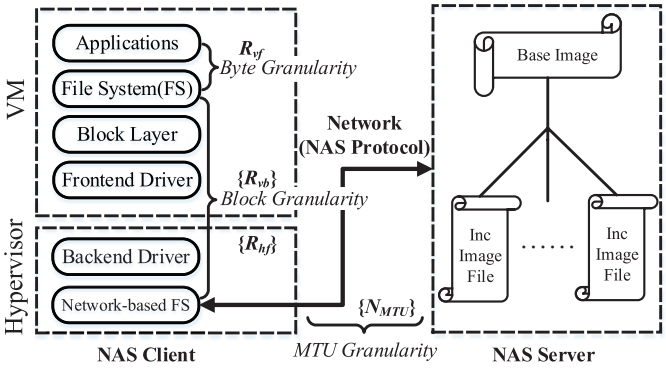

Fig. 2. Virtualized NAS I/O stack. Even a CPS-based application reading/writing a few of bytes from/to the underlying file system, the combination of VM and NAS will complicate and deepen the I/O path of a simple system call which derives from a VM's application.

1.2 Analysis and Background

According to different configurations, Table 1 depicts the VM startup time as well as the amount of transferred data over NFSv3 protocol. `1-VM-DAS`, in which only one VM interacting with local disk, acts as the baseline to compare with the NAS ones. For `1-VM-NAS-1-Client`, where only one VM connecting with the NAS server, spends about 42 seconds to finish the startup procedure, without any interventions from another VMs. It achieves 71.4% performance of `1-VM-DAS` due to the extra network delay between VM and its persistent data. It should be noted that about 11MB data is added into the network transferring, from 231MB to 240MB, on account of the data encapsulation along NFSv3 protocol stack, as Figure 2 illustrating.

Five co-located VMs boot simultaneously in the case of `5-VMs-NAS-1-Client`. Since only one client is used in this scenario, the procedure of VM startup, a read-intensive workload, forms a performance bottleneck in the server's network adapter, thus congesting the data sender. In this case, end-users only gains only 32.6% performance of `1-VM-DAS` and feels an explicit stagger subjectively. Finally, `25-VMs-NAS-5-Clients` aggravates this congestion by multiplying the number of VM from 5 to 25, in which each client hosts 5 running VMs. The NAS connections between server and client are overwhelmed by these booming data

requests. On average, one end-user waits up to 380 second, which is 13 times longer than `1-VM-DAS`, to completely acquire his virtual desktop. This problem, being termed as *boot storm*, causes the interaction of virtual desktop slow and unresponsive, even makes a VDI-based CPS project unpopular among end-users.

TABLE 1
Time spent and the amount of data transferred over NFSv3 protocol during VM startup. The VM startup time is the sum of VM booting, user logging and system initialization. We use the mean of 5 trials by recording the average time of all end-users launching their virtual desktops.

Configuration	VM Startup Time (Seconds)	Data Transferred (MB)
1-VM-DAS-1-Client	30	231
1-VM-NAS-1-Client	42	240
5-VMs-NAS-1-Client	92	1,267
25-VMs-NAS-5-Clients	388	6,320

Unfortunately, in addition to *boot storm*, there are quite a few similar issues in current VM-based cloud infrastructures. For example, *auto-update* routine in user VM is usually being triggered at the fixed time in a day. At cluster-scale, this operation perhaps spawns a lot of concurrent write-intensive workloads (a *read-modify-write* style) to flood the underlying NAS connection, resulting in the poor VM disk I/O performance. *Swapping*, for another example, needs to exchange pages from disk to memory when the system memory runs out. NAS deepens this swapping path in light of the extra network delay, and a NAS congestion would further deteriorate the VM's swapping velocity.

In summary, with a large amount of live VMs, how to effectively improve the VM disk I/O performance in a congested NAS environment has been a big challenge. Upgrading network equipment, such as migrating a 1Gb network to the 10Gb one, is a potential solution to relieve above issues, but it comes at cost of huge investment on hardware. Only taking switch as an example, Cisco 10Gb switches are up to about 18 times (on average) more expensive than the 1Gb ones according to the latest price list on *Amazon.com* [16]. Therefore, there is tradeoff here between VM disk I/O performance and hardware investment.

1.3 Our Contributions

In this paper, we propose a *software-based* method, Triple-L, to improve the VM disk I/O performance in a typical CCPS environment. Triple-L decouples VM image file at different storage semantics and then distributes these sub-files into the NAS clients, where is usually capable of the persistent storage to act as the local NAS cache. In this way, instead of experiencing the NAS connection with uncertain performance, VM disk I/O requests who targeted at these sub-files can be served locally, thus walking the *shorter* and *wider* storage path than NAS.

Specifically, Triple-L includes three sub-optimizations:

- *Shadow-base*, transfers the VM base image into NAS client. When VM issues read requests into the base image file, they can be satisfied locally, thus improving their performance while decreasing the amount of network transmission from NAS server to clients. This mechanism is quite useful in the event of *boot storm*.
- *Log-Split*, redirects VM's journal block device (JBD) into NAS client. By extra tuning on the journaling mode, the completion of VM write can be asynchronized to the checkpoint routine of this local JBD, thus eliminating the time overhead for transmitting VM's dirty data over NAS connection.
- *Local-Swap*, decouples virtualized swapping device from VM's image file and then re-locates swap I/O into the local disk. So every swapping operation can be proceeded in the NAS client, instead of delivering out to NAS server.

We have implemented aforesaid optimizations in a specific virtualized system based on NFSv3. An initial solution to storage failure as well as VM live migration is also presented or discussed. The experimental result shows that our methods can avoid up to 77.6% network traffic when multiple VMs issue disk I/O requests simultaneously under a 1Gb network. In the mean time, the performance of VM disk I/O achieves 17%-82% improvement in different experimental scenarios.

The rest of this paper is organized as follows. We present the design of our methods in section 2 while discussing the potential downside and its preliminary improvement. Section 3 shows the implementation of our method on a specific virtualized cloud system. Section 4 describes the experimental methodology and discusses the result. In section 5, we discuss the related work. Finally, section 6 concludes the paper and talks about the future work.

2 Design

In this section, we first describe the sub-optimizations in Triple-L in turn, and then give a discussion about their potential side effects upon system failure and VM live migration. The initial solutions to them are also presented.

2.1 Shadow-Base

For rapid VM provisioning and snapshotting, many cloud infrastructures use QCOW-like image format to organize the VM's disk data on the side of NAS server. As the right side of figure 2 showing, a read-only base image file, usually containing the minimum file system of VM, is shared among VMs' private virtual disks. Each private virtual disk of VM is an incremental image file to the base image and be allocated by the strategy of copy on write (COW). According to the documented details of QCOW2 [17], VM's write request, excluding rewrite one, appends new data to the end of an incremental image file. In doing so, VM's physical disk data can be allocated on the demand of end user.

In the procedure of bootstrap, VM reads a certain amount of files from the base image in NAS server, including the necessary kernel information to boot an OS and some applications which are pre-installed to the particular users. Therefore, when many VMs boot in a short time, a batch of read-intensive workloads targeting on the base image will form *boot storm*. This motivates our first improvement, called *Shadow-Base*, puts a replica of the base image into

NAS client, allowing the read traffic on the base image to be isolated into the local file system of VM. Intuitively, this DAS/NAS hybrid architecture could substantially reduce the network traffic on $\{N_{MTU}\}$[1] in the event of *boot storm*, while improving the responsiveness of a certain part of $\{R_{hf}\}$. Another similar usage scenario of *Shadow-Base* is *anti-virus storm*, which happens when a significant number of VMs are scheduled to run malware scans at the same time. This activity also involves read-intensive workloads focused on the base image, where a lot of system-critical files are stored.

With *Shadow-Base*, NAS server needs to distribute the replicas of base image into every NAS client. To avoid the network rush hour, this work is often proceeded at the free time, such as weekends and holidays. However, an emergency update to the base image, such as fixing vulnerability in user VM, introduces a challenge of quick replica updating, which is required to be addressed in a short time and to cover all base image files across the related NAS clients.

We improve this issue by a technique called *Inplace-Update*. When a VM is alive, any updates or rewrites upon its base image file (denoted by B_C) are only reflected on one *particular* and *local* incremental image file (denoted by I_C), which stores in NAS client to avoid the NAS connection. Meanwhile, the original base image file in NAS server, denoted by B_S, is immediately patched with the intervention of system administrator when new patches are ready. Correspondingly, a particular incremental image file, I_S, is created to apply this update into B_S. Then, when the whole system has came at the free period, *Inplace-Update* makes two new base image files, B_{NC} and B_{NS}, by combining I_C, I_S into B_C, B_S respectively, on either side of NAS client and server. During this procedure, *Inplace-Update* will check the consistency between B_{NC} and B_{NS} by using the `SHA-1` hash function [18]. If a difference is detected, B_{NC} will be replaced with B_{NS}, to ensure that the update operations applied on B_C only derives from the procedure of VM patching.

2.2 Log-Split

With `copy-on-write` manner, an ordinary VM write request overwrites or appends the incremental image file in NAS server. For example, when a user in one VM installs his favorite personal software, the associated data will be persisted by the incremental image file at host layer. It should be noted that this kind of VM write is different from the update/rewrite one which targeted on the base image file in *Inplace-Update*. They are all required to traverse the network path and aim to the incremental image file. Unfortunately, journaling (or logging), which is widely used in modern file systems, slows this write procedure by the characteristic of *double writes* [19].

In general, journaling file system persists the writing data twice in the underlying storage device, one is in the journal area, the other is in the main file system. In such a way, by checking the small and dedicated journal area, the file system can be brought back on-line quickly and less likely become corrupted in the event of a power failure or system crash [20]. However, the strict serialization of *double writes* would defer the completion of an ordinary VM write request in NAS. For example, an application in one VM issues an ordinary write request. With *ordered-mode* in `ext3` file system, (i) the application must wait until the regular data to be successfully transferred into the main file system in NAS server and then (ii) wait the related meta-data to be persisted in the journal area over NAS connection again. Therefore, if this ordinary VM write request is explicitly synchronized from end-users, a delay may be perceived not only due to the obvious network path but also due to the uncertain performance on NAS connection.

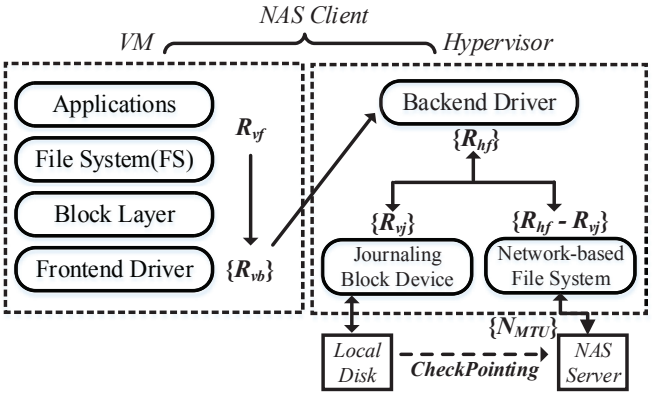

Fig. 3. Illustration of Log-Split. The logging data is decoupled from the block-level VM disk I/O requests and then being redirected into the local disk device. If a certain condition is true, such as the journal area is full or a timer is expired, the checkpointing resided in the VM journal block device will be triggered to synchronize the local journal data into NAS server.

To improve VM write performance, we propose the second sub-optimization called *Log-Split*. As figure 3 illustrating, *Log-Split* identifies the semantic of VM logging in $\{R_{hf}\}$, allowing NAS client to treat the VM logging flows separately. In detail, *Log-Split* extracts the VM logging flow, denoted by $\{R_{vj}\}$, from $\{R_{hf}\}$, and then redirects them into the local block device on the side of NAS client. In doing so, the transmission of VM logging data is asynchronized from $\{R_{hf}\}$, thus improves the logging performance as well as reduces the amount of $\{N_{MTU}\}$ delivered over NAS connection. [2]

Eventually, the logging data stored in NAS client will be transferred into NAS server by the checkpointing routine of local journaling block device (JBD), both for recycling the space of journal area and for a consistent VM image file. However, there is a potential benefit hazard in the case of VM sustaining a write-intensive workload, in which the checkpointing routine will be frequently invoked to clean the journal space, thus blocking the incoming logging requests. We treat this result as a justified feedback from the system, since an I/O intensive VM should accept the performance punishment for its irregular behavior. But *Log-Split* needs to enforce an isolation mechanism on the local JBD, preventing the co-located VMs from being disturbed. Currently, we simply set a fixed journal size for each VM,

1. The symbol is defined in Figure 2.

2. An ordinary write request of a VM can even be completed locally with *Log-Split* under some particular logging modes, such as *journal-mode* in `Ext4`, not only for the logging request.

while adding the FIFO (First-In-First-Out) queue on each journal area with fixed size too.

2.3 Local-Swap

Since the limited amount of memory are sharing among co-located VMs, swapping is necessary in a virtualized cloud system [8]. This process exchanges the allocated but inactive VM memory pages out into disk and makes room for those being accessed frequently ones. In NAS, the memory-intensive workloads from multiple VMs, whose demand for memory resource exceeds the total size of physical memory, must experience the *longer* and *narrower* swapping path as figure 2 showing. For example, one memory page in a certain VM, denote by p, is being *swapped out* from VM memory. Since the swap partition or file is mapped in the VM incremental image file, p should walk the NAS connection to contact with NAS server if it missed the swap cache, thus incurring network delay. Moreover, an uncertain performance provided by the jammed NAS connection would further slow this procedure. The same situation may also exist in the case of p being *swapped in*. Therefore, consider the memory operation is critical to a typical OS, the swapping performance of VM is challenged in the presence of NAS storage architecture.

We propose the third sub-optimization in Triple-L, called *Local-Swap*, to improve the swapping performance of a virtualized NAS architecture. The idea behind *Local-Swap* is similar with *Log-Split*. We decouple $\{R_{vs}\}$, the disk I/O of VM swapping, from $\{R_{hf}\}$ and then isolate the VM swapping I/O into the local disk device of NAS client. By so doing, all swapping I/Os from VMs will be handled locally, eliminating the corresponding network traffic as well as the potential jammed NAS connection.

Local-Swap can cooperate well with the memory balloon driver in hypervisor [21]. By sampling the velocity of swapping I/O efficiently in an inflating VM, our sub-optimization over a virtualized NAS system can improve the reaction speed of balloon driver, allowing this inflated VM to be deflated quickly without a significant performance degradation. On the other hand, *Local-Swap* is orthogonal to the other sub-optimizations in Triple-L. In fact, some special VM workloads can even obtain the resultant benefit from all of them. For example, co-located VMs in a NAS client are launched simultaneously (*Shadow-Base*). Then, due to physical memory pressure in this host, hypervisor swaps any pages out into backing storage (*Local-Swap*). Meanwhile, system log is recording the information of user logging in and thus produces several write requests to its file system (*Log-Split*).

2.4 Discussion

This subsection answers the following two questions: (1) *How does Triple-L respond to storage failure?* (2) *How does Triple-L handle with VM live migration?*

2.4.1 Storage Failure

From a coarse-grained taxonomy, storage failure in Triple-L may happen at two places: (i) NAS client and (ii) NAS server. Upon NAS client failure, Triple-L may lead a portion of CPS data to be lost due to its local cache. Specifically, *Log-Split* suffers such a loss in that the logging data may have not been synchronized into NAS server once failure appeared. Therefore, this sub-optimization on VM write performance delivers a recovery point objective (RPO) of non-zero (i.e., non-zero data loss) [22]. To deal with this issue, Triple-L explicitly reminds users when a new VM is allocating to them, giving user an obvious option to activate *Log-Split* or not. On the other hand, *Shadow-Base* is capable of finding these data back in light of *Inplace-Update* in NAS server. *Local-Swap* could also ignore this kind of loss on VM swapping data, who have not being flushed into backing storage, because the ordinary OS has treated data in DRAM as unreliable.

When the storage of NAS server confronted with a failure, Triple-L can only restart a VM from the base image file persisted by NAS client with *Shadow-Base*. The data created in the incremental image file in NAS server may be lost. They are only able to being restored by the inherent backup mechanism in NAS server such as deploying RAID 1 system.

In summary, compared with the original NAS architecture, *Log-Split* in Triple-L trades off the reliability of VM logging data, perhaps leading a portion of user data to be lost. VMs that cannot risk any data loss should disable *Log-Split* or apply another enhancement techniques into the storage reliability of NAS client.

2.4.2 VM Live Migration

Compared with the original NAS architecture, Triple-L complicates the procedure of VM live migration [23] in that local sub-files in *Log-Split* and *Local-Swap* are both required to transferred into the destination host, except for *Shadow-Base*, which avoids this procedure by virtue of the replica of VM base image file existing in NAS Server.

`Algorithm 1` describes the detail procedure of Triple-L handling with VM live migration. Generally, Triple-L uses memory balloon driver to deflate this migrating VM, allowing the size of active swapping area to be become as small as possible. Then, the `pre-copy` technique is used to transferring the memory data of VM. This process firstly sets all pages of the VM as dirty, then successively scans VM memory and copies dirty ones from source to destination repetitiously [24]. When the remaining dirty pages to be copied is smaller than a pre-defined value, which would not produce the obvious VM downtime to user, journaling block device in source VM triggers the checkpointing routine to archived the committed logging transactions to NAS server thus reducing the size of active ones. After that, the VM execution is suspended. At this time, both data of active logs and swap areas are copied, and the remaining dirty pages are also sent to the destination completely. Finally, the fully consistent VM on the destination is resumed. Meanwhile, the source VM is killed.

Aside from deflation and checkpointing, `LZ4` compression algorithm [25] [26] is applied into the copying procedure of active logging and swapping data, further decreasing the underlying network traffic to lively migrate a running VM. Besides, Triple-L also supports the feature of VM snapshot [27] with a similar solution. When a user or system administrator explicitly invokes the snapshot

Algorithm 1: Description of Triple-L Handling with VM Live Migration

Input: External event triggers VM live migration
Output: Success (true) or failure (false)

```
m = false;
/*m is a flag to determine if the remaining dirty pages of VM are small enough.*/;
Deflate_VM(void);
while (!m) do
    PreCopy(Memory, m);
end
Check_Pointing(Journal);
Pause_VM_at_Source(void);
Copy_to_Destination(ActiveLog,ActiveSwap);
MigratingVM(void);
Restore_VM_at_Destination(void);
return (true);
```

interface, deflation on VM memory as well as checkpointing on logging data will also be applied.

3 Implementation

We implement all above three sub-optimizations in a NAS architecture based on NFSv3 protocol. The NAS client is virtualized by Xen hypervisor.

To *Shadow-Base*, we first put the replica of a base image into the local disk device of NAS client. Then we modify the VM boot configuration, allowing the path of base image to point to the local replica, instead of the NFS partition mounted on NFS client. A block-level filter in Xen `blktap2` is interposed to the `QCOW2` path, allowing the write request targeted on base image file to be redirected into the local backing storage (*Inplace-Update*).

To *Log-Split*, we pass an extra physical block device (denoted by B_j), which built in the local disk device of NAS client, into the VM. By using the Xen split driver model, VM can identify B_j and format it with the suitable file system such as `Ext2`. Then we move the VM's journal file into B_j. The reason for the choice of `Ext2` is the unnecessary of double journaling inside a VM [28] as well as the small size of B_j (e.g. 16MB). An additional refinement for boosting the concurrent writing performance of co-located VMs in a NAS client is coalescing different B_js into a gathered storage area in the host layer. In doing so, the local disk device of NAS client can receive a more sequential I/O flows thus improving its throughput.

Local-Swap shares the similar implementation with *Log-Split*. On the side of NAS client, we first allow a dedicated physical block device in host layer (denoted by B_s) to be visible in a VM, and then create a new swap partition up on B_s. This work is often done when VM is off-line. After this configuration, the page fault from VM will be handled locally atop B_s. Due to the large size of B_s (e.g. 1GB), which is usually recommended as the double size of a VM memory, *Local-Swap* would not merge the swap devices among co-located VMs for the negligible benefit.

To VM live migration on Triple-L, we first use `xm mem-set MEMORY_UPPERLIMIT` command to replenish the available memory resource into this migrating VM, triggering its swapped data to be paged in. Then, we leverage `ioctl()` to devise an interface on the JBD of VM, allowing the `log_do_checkpoint()` routine to be invoked explicitly to checkpoint the underlying logging data. Finally, `xm migrate` command is used for migrating this VM, in which pre-copy technique has been built-in to copy dirty pages in rounds. Note that a simple copy process which transferring the VM active logging and swapping data to destination, is interposed between the two time points of VM pause and resume.

4 Evaluation

This section presents the experimental results in five scenarios, including (1) boot storm simulation, (2) write-intensive workload, (3) memory consumption test, (4) virtual-desktop responsiveness and (5) VM live migration. The prior three scenarios are independently evaluating the effect of *Shadow-Base*, *Log-Split* and *Local-Swap* in turn. The two other tests are used to check the comprehensive influence of Triple-L on the realistic desktop workloads and VM live migration respectively. We use five NAS clients and one NAS server to constitute the experimental platform. All detail configurations have described in section I.A.

4.1 Boot Storm

In this scenario, 25 VMs, in which each NAS client hosts 5 VMs, will be cold-booted simultaneously. At first, we measure the local boot time of a single VM to acted as the performance baseline. After an experiment with three rounds, we find that a single VM bootstrap from the local disk device (NAS client) will last about 30 seconds on average, in which further divides into three phases: VM booting, system initialization and user logging. The prior two phases reads about 163MB data from the underlying local disk device while the last one reading about 69MB.

Figure 4 shows the result. Under the original NAS architecture, approximately 6.3GB data is transferred via the 1Gb network. The startup time of each VM is 388s on average, increasing the time delay by about 13 times compared with the baseline, from 30 to 388. We find the network utilization on the NAS sever keeps saturated throughout the *boot storm* period. Therefore, this cost is partly due to the network transmission from NAS server to client and then be amplified by the network congestion. In addition, a local contention of the shared disk device in NAS server deteriorates this situation, since up to 25 I/O threads are established to concurrently read the same base image.

Shadow-Base achieves 77.6% reduction of network traffic load from 6.3GB to only 1.68GB, by distributing replicas of the base image among NAS clients. The rest of network traffic derives from the disk I/O of user logging, whose block data is user-specific and thus be often stored in the incremental image file in NAS server. On the other hand, the VM startup time is also improved by 81.7% from 388s to 71s. This benefit is mainly due to the short data path of base image, where is located in the NAS client. Compared

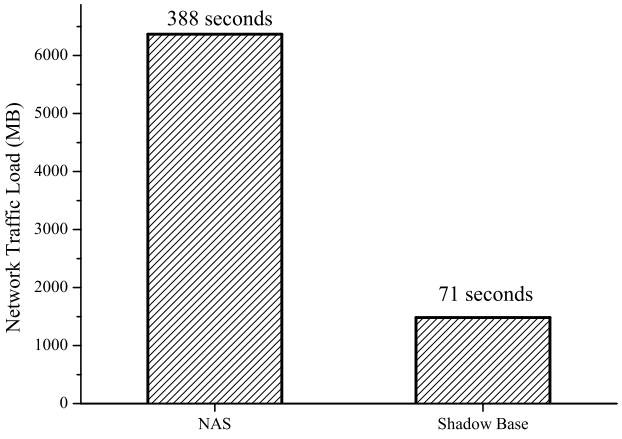

Fig. 4. Network traffic load on NAS server in the case of boot storm. The numeric value atop the bar is the startup time of each VM on average.

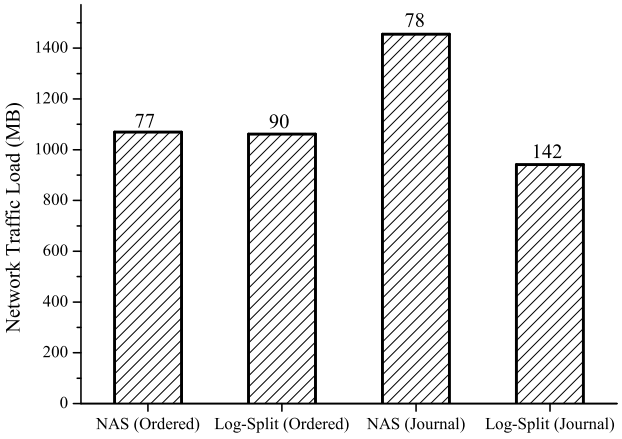

Fig. 5. Network traffic load on NAS server in the case of write-intensive flow. The numeric value atop the bar is the IOPS (Input/Output Operations Per Second) of each VM on average.

with the original NAS architecture, fewer VMs, from 25 to 5, share the same base image on the local disk device, relieving the resource contention of reading the base image. This is another contributor of *Shadow-Base* to boost the performance of VM startup.

We also simulate a scenario to test the effect of *Inplace-Update*. Firstly, in every VM we prepare `Ubuntu Desktop 13.10` release to end user. Then, we explicitly set each VM to download the newer upgrade package (14.04 release) via `Update Manager`. The package size for each VM is around 800MB. When this data has been ready, we launch all upgrade procedures of 25 VMs concurrently, by using `do-release-upgrade` command with `shell` script. Table 2 gives the result.

TABLE 2
Time elapse and network traffic load during VM upgrade (Discounting the download phase). We use the mean of 5 trials by recording the average time of each VM. Baseline means the case that a VM runs on the DAS architecture.

Configurations	Upgrade Time (Minutes)	Total Network Traffic (MB)
Baseline (1 VM)	46	566
NAS (25 VMs)	116	9,049
Triple-L (25 VMs)	62	4,542

As baseline, a VM takes 46 minutes to complete its upgrade procedure, while 566MB data being written to the local backing storage. Note that there is a difference existed between the size of upgrade package (800MB) and the actual writing data (566MB), since a portion of software packages were the latest version in this VM. For the original NAS, VMs spend about 2 hours to finish this work on average, which is 2.6 times longer than the baseline. Meanwhile, approximately 9GB data are transfered over NAS connection and be written to NAS server. Triple-L reduces this time overhead by 46.5% from 116 to 62. There is 4.5GB data still be transferred to NAS server, but saved by 49.8% from 9,049 to 4,542. The transferred data via NAS connection derives from the older user-specific applications, which are stored in NAS server and thus be updated in this *upgrade storm*.

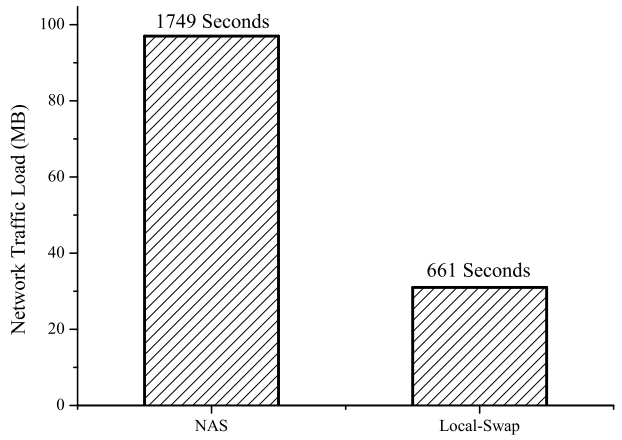

Fig. 6. Network traffic load on NAS server when VMs running memory intensive workloads. The numeric value atop the bar is the average benchmark runtime of each VM.

4.2 Write Intensive Workload

In this scenario, we use the benchmark `IOmeter` [29] to test the effect of *Split-Log*. The testing file is 1GB and each I/O operation issued by `IOmeter` is 4KB in target size with 100% random write operation. The journaling size is set to 128MB for each VM.

Figure 5 shows the result. In the *ordered* journal mode of VM file system, *Log-Split* transfers the almost identical network traffic with the original NAS architecture. We conclude this result is due to the specific behavior of *ordered* journal mode, although the metadata is immediately logged in the journal area of NAS client. However, this cached metadata in NAS client is required to move to the NAS server later, thus finally reaping the almost same network traffic with original NAS. Note that the procedure is asynchronized from the path of VM normal write by *checkpointing* inside VM's journaling block device (JBD). Therefore, *Log-Split* still improves the IOPS of VM by 17%, from 77 to 90. In other words, the performance gains are due to the local logging of VM metadata, allowing a part of VM write requests to be finished in NAS client.

In the *journal* mode, *Log-Split* reaps 35.3% reduction

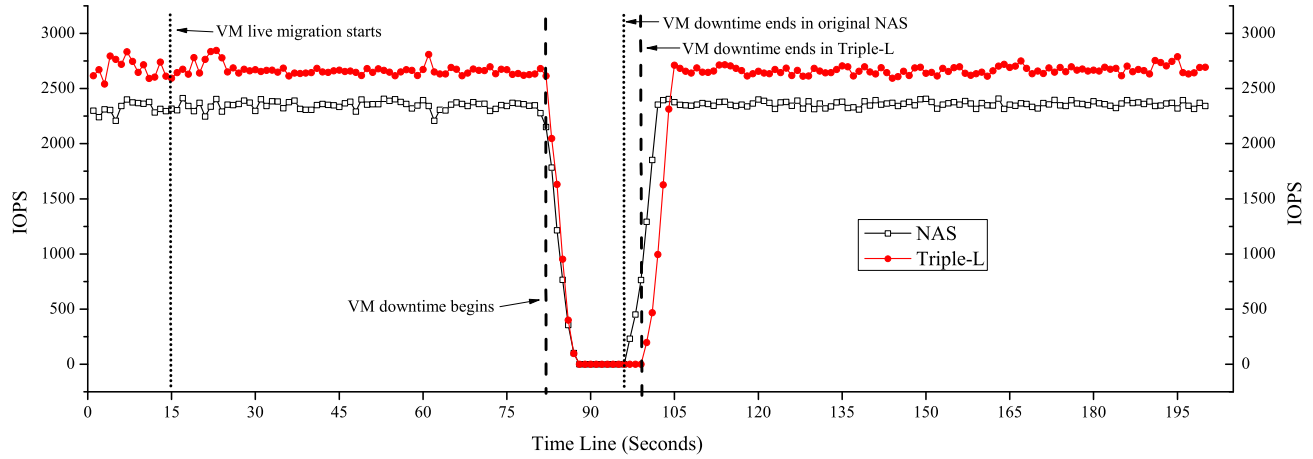

Fig. 8. Variations of IOPS during a single VM live migration. The available network bandwidth for VM live migration is set to 250Mbps to simulate a jammed condition.

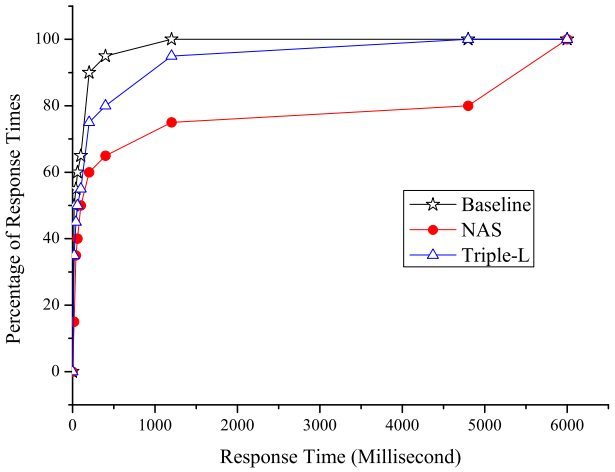

Fig. 7. CDF plot of the average interactive response times of 25 VMs in NAS client. The response time is measured by the time difference between the event of user input (e.g. user clicking a menu entry) and the corresponding change on current desktop environment (e.g. a dialog box being popped up). The case of baseline means a single VM running the recorded trace under DAS architecture.

effect of network traffic load on NAS server compared to the original NAS architecture. This improvement derives from that JBD with *journal* mode can log both metadata and data locally, in which absorbs a part of VM write as rewrite, thus eliminating their network transmissions during a short time window. On the other hand, *Log-Split* with *journal* mode surprisingly archives 82% improvement from 78 to 142 in VM IOPS compared to the original NAS. It has even the 58% improvement over the *Log-Split* with *ordered* mode, from 90 to 142. We conclude this benefit comes from that `fsync()` issued by VM application under the *journal* mode can be finished when the metadata and the data are once logged into the local JBD, rather than the *ordered* one, in which `fsync()` must wait until the data have been transferred into NAS server.

4.3 Memory Intensive Workload

Java faces a challenge in a virtualized system as its garbage collector causes a pathological situation of degraded performance when the physical memory allocated to the guest is smaller than the Java virtual machine (JVM) working set. Therefore, in this scenario we use the `Eclipse` workloads that are part of the `DaCapo` Java benchmark suite [30] to produce the memory intensive workload to test the effect of *Local-Swap*. The memory size of all VMs in this experimental scenario are set to 128MB. All 25 VMs use OpenJDK and a 128MB heap. In doing so, we can lead the VM to frequently swap in/out its memory pages.

Figure 6 depicts the benchmark result. Clearly, *Local-Swap* achieves the reduction of network traffic by 68%, from 97MB to 31MB. The remainder of network activities are aroused by the disk I/O of VM running DeCapo Java benchmark. Similarly, a 62.2% performance benefit, from 1,749 seconds to 661 seconds, is obtained due to the short path of VM swapping, where is only experienced inside NAS client.

4.4 Realistic Virtual-Desktop Workload

Here we evaluate the responsiveness of a practical virtual-desktop workload on Triple-L in which the logging manner in *Log-Split* is set to `journal` mode. We first use `VNCPlay` [31], a cross-platform tool for measuring interactive performance of GUI-based systems, to record a group of user interactive behaviors on a VM `GNOME3` desktop. The user-behavior includes a huge amount of user desktop operations, such as editing documents by `Eclipse-Luna-SR1 (4.4.1)` and browsing web-pages by `Firefox 33`. Then, we replay this recorded trace inside all 25 VMs in NAS client, which are sequenced by following the `Poisson` distribution [32]. Therefore, a mixed workload consisted of memory- and write-intensive operations is produced.

Figure 7 shows the average result among 25 VMs. Compared to baseline, Triple-L incurs a moderate performance punishment, in which the user operations exceeded 200ms accounting for 25% (only 10% in baseline). We conjecture this overhead mainly comes from NAS connection due to

the user-specific applications being stored in NAS server, thus confronting with the longer response time on their operations. On the other hand, user experience in the case of original NAS is deteriorated sharply. More than 40% user operations exceed 200ms. Of these slow operations, there are still 50% user operations even incurred at least 4,800 millisecond delay, which could lead to an explicit stagger during user interacting with his desktop environment. Since the write operations of VM were finished quickly in NAS client, Triple-L eliminates most of this kind of delay. Moreover, a local swapping procedure allows the VMs with memory pressure to handle their page faults locally, thus further accelerates the response time of virtual desktop.

4.5 VM Live Migration

Compared with the original NAS architecture, Triple-L needs to copy extra disk data from source host to the destination in VM live migration. This scenario is devised to test this influence brought by Triple-L. We first use `Iometer` to enforce a sustained I/O flow on a single VM. The testing file is 1GB and each I/O operation issued by `Iometer` is 4KB in size with 100% random write operation. Meanwhile, we also replay the recored user trace which has collected in section IV.D in this VM. Therefore, we ensure that there is an active logging data as well as the swapping area existed in the migrating VM.

Figure 8 depicts the variations of IOPS along a single VM live migration. Triple-L achieves 77.8% migration performance against the original NAS from 14s to 18s to finish the underlying VM down time. It concludes that our methods, namely checkpointing and deflation before the phase of VM `stop-and-copy`, incur a moderate overhead for copying the extra logging and swapping data. We speculate that this overhead mainly derives from the active transaction in JBD under the sustained write flow. A potential improvement on this downside is using the pre-copy technique over JBD in NAS client. We will verify this point in our future work. On the other hand, Triple-L has a 12.8% performance advantage over the original NAS in IOPS if excluding the phase of VM downtime. The reason is that Triple-L uses *Log-Split* with `journal` mode to improve the write path of VM. But this benefit is limited by the fixed space of each VM, namely 128MB, leading checkpoint routine to be frequently invoked to clean space while blocking the incoming write requests.

5 Related Work

Currently, there are some existed works which have explored the performance problem of concurrent VMs sending disk I/O requests over NAS networking. These works can also be applied into a CPS system.

Chen et al propose a BitTorrent-like P2P protocol to accelerate the provisioning of VM image files by leveraging the NAS clients' upload capacity [33], thus improving the traditional bandwidth bottleneck of NAS server. Shadow-Base further improves this method by absorbing a lot of network traffic which targeted on the NAS client, rather than consuming any NAS clients' upload links.

Peng et al present VDN [34], a chunk-level topology-aware collaborative sharing for distributing VM image files. This technique is based on the observation that the conventional P2P sharing strategy may not be applicable since the number of same VM instances is not big. VDN differs from Triple-L in that VDN only accelerates VM read operations while Triple-L improves both directions of VM read and write and swapping I/O.

Nicolae et al introduce a lazy VM provision mechanism that acquires VMs' target I/O data as demanded by the application running in the VM [35], to ease the pressure on the VM storage for heavily concurrent disk I/O requests. Meanwhile, an improvement against VM snapshot is presented in their works. Triple-L does not involve VM snapshot, but focuses on improving VM performance of launching, logging and swapping. Shadow-Base technique in Triple-L completely isolates VM disk I/O from NAS networking, instead of a lazy fetching scheme which still produce the uncertainty of network traffic.

Reich et al propose VMTorrent [36] to enable effective VM image streaming. Compared with Triple-L, this work mainly aims to quickly launch a single fresh VM and load its critical applications. It differs from Shadow-Base in that VMTorrent will spawn a lot of network traffic when any VMs want to acquire the booting data. Instead, by the local persistent cache where stored the VM base image, Shadow-Base may eliminate this data exchange between NAS client and server.

Capo [37] reduces disk I/O traffic load by using local disk as persistent cache, using a preloading technique to broadcast read results across a cluster, and by imposing differential durability to divide a VM's file system into regions of varing writeback frequency. Our work shares the similar idea with Capo, but differs from it in that: (i) *Shadow-Base* shares a base image inside every host (NAS client) and thus eliminates the network traffic of Capo broadcasting the associated data about base image across hosts; (ii) Capo divides VM image file into more pieces by leveraging the semantic of OS-dependent directories. This restricts Capo to be a single solution focusing on the Windows-based VM, complicating its implementation across various OSes. Instead, our design overcomes this handicap by using a more high-level abstraction to organize VM image file. This feature makes our design more generic among certain specific systems.

FVD [38] is an enhanced image format of QEMU. To reduce the load on storage server, it embeds three features, namely copy-on-write, copy-on-read and adaptive prefetching, into the VM image file. Compared with our work, FVD improves `QCOW2` at the host file system, without involving any interior semantic of VM, thus it can be combined with our three sub-optimizations as a complementary method.

Tarasov et al [9] study the transformation of existing NAS I/O workloads due to server virtualization. They find great changes, including the disappearance of file system meta-data operations at the NAS layer, not only varies the I/O size, but increases randomness. Based on the observations, they develop new benchmarks that correctly represent NAS workloads in a virtualized data centers. This work provides insight into the virtualized NAS architecture and motivates our improvements.

On the host-side flash-based cache, Koller et al develop two write caching policies in the environment of

networked storage to strike new and useful tradeoffs across performance, consistency and staleness dimensions [22]. This work focuses on the block-level VM requests at host level, irrespective of the specific VM interior semantic. It offers opportunities to our work for mutual enrichment and cooperation. For example, these write policies can wrap our *Split-Log* and provide an option of the balance between the reliability of VM write and its performance.

By decoupling the control of IO flows from the data plane, IOFlow [39] can offer flexible service and routing properties in data centers, thus is capable of applying the end-to-end policy into the I/O path from VM (NAS client) to the shared storage (NAS server). hClock [40] is another similar work to IOFlow. It is a hierarchical bandwidth allocation algorithm and provides rich QoS (Quality of Service) controls for network allocation in a hypervisor. Thus, hClock can also be used to handle the disk I/O of VM in a cloud platform. More recently, through a combination of priority and token-bucket rate-limiting, Zhu et al presents PriorityMeister [41], a proactive QoS system that attains end-to-end tail latency SLOs (Service Level Objectives) across multiple stages. IOFlow, hClock and PriorityMeister are complemented to Triple-L since the end-to-end policy can preserve the available bandwidth to the specific VM and even to the internal storage semantic, such as the checkpointing flow in *Log-Split*, thus further improving the VM disk I/O performance.

6 Conclusion

In a CCPS environment with virtualized NAS backing storage, the combination of VM and NAS protocol has the advantage of reducing costs and simplifying management, but a significant drop in the performance of VM disk I/O is observed due to the introduction of complicated storage layers, including the network delay between NAS server and client. We propose a set of optimizations, which are designed around the high-level semantic carried in the VM file system, to improve such a performance bottleneck. We use three high-level semantics to organize the VM image file, namely (i) the minimum and common VM file system (VM base image), (ii) JBD and (iii) swapping area. By breaking down the original VM image file into several pieces according to these semantics, a portion of disk I/O would be isolated in the local file system of NAS client. Then, we use typical experimental scenarios to verify our design and it shows a substantial improvement. Finally, we also devise a specific scenario to test the influence on the VM live migration. The result is also promising.

Acknowledgments

This work is supported by the following projects: Project funded by China Postdoctoral Science Foundation (No. 2014M552214); The PhD Start-up Fund of Natural Science Foundation of Guangdong Province of China (Grant No. 2014A030310238); CCF-Tencent Open Fund (Grant No. CCF-TencentIAGR20140102), JSPS KAKENHI Grant No. 26730056, 15K15976, JSPS A3 Foresight Program; National High-Technology Research and Development Program (863 Program) of China (Grant No. 2013AA01A212); National Natural Science Foundation of China (Grant No. 61272067, 61370178); Natural Science Foundation of Guangdong Province of China (Grant No. S2012030006242); Science and Technology Support Program of Guangdong Province of China (Grant No. 2012A080104019, 2011B080100031); MOE-China Mobile Research Fund (No. MCM20121051); Guangzhou Research Infrastructure Development Fund (No. 2012224-12); Guangzhou Zhujiang Science and Technology Future Fellow Fund (No. 2011J2200089).

References

[1] H. Li, P. Li, S. Guo, X. Liao, and H. Jin, "Modeap: Moving Desktop Application to Mobile Cloud Service," *Mob. Netw. Appl.*, vol. 19, no. 4, pp. 563–571, Aug. 2014.

[2] M. Dong, K. Ota, X. Li, X. Shen, S. Guo, and M. Guo, "HARVEST: A Task-objective Efficient Data Collection Scheme in Wireless Sensor and Actor Networks," in *Proceedings of the 3th International Conference on Communications and Mobile Computing (CMC 2011)*, April 2011, pp. 485–488.

[3] S. Mills, "Putting the Classroom in the Cloud with Virtual Desktops and Bring-Your-Own-Device," *eLearn*, vol. 2014, no. 4, Apr. 2014.

[4] M. Dong, K. Ota, H. Li, S. Du, H. Zhu, and S. Guo, "Rendezvous: towards fast event detecting in wireless sensor and actor networks," *Computing*, vol. 96, no. 10, pp. 995–1010, 2014.

[5] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov, "OSv—Optimizing the Operating System for Virtual Machines," in *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014.

[6] J. Ouyang and J. R. Lange, "Preemptable Ticket Spinlocks: Improving Consolidated Performance in the Cloud," in *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2013)*. New York, NY, USA: ACM, 2013, pp. 191–200.

[7] M. Dong, K. Ota, M. Lin, Z. Tang, S. Du, and H. Zhu, "UAV-assisted data gathering in wireless sensor networks," *The Journal of Supercomputing*, vol. 70, no. 3, pp. 1142–1155, 2014.

[8] O. Agmon Ben-Yehuda, E. Posener, M. Ben-Yehuda, A. Schuster, and A. Mu'alem, "Ginseng: Market-driven Memory Allocation," in *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2014)*. New York, NY, USA: ACM, 2014, pp. 41–52.

[9] V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok, "Virtual Machine Workloads: The Case for New Benchmarks for NAS," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 2013)*. Berkeley, CA, USA: USENIX Association, 2013, pp. 307–320.

[10] R. Birke, M. Bjoerkqvist, L. Y. Chen, E. Smirni, and T. Engbersen, "(Big)Data in a Virtualized World: Volume, Velocity, and Variety in Cloud Datacenters," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 2014)*. Santa Clara, CA: USENIX, 2014, pp. 177–189.

[11] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, "XvMotion: Unified Virtual Machine Migration over Long Distance," in *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014.

[12] S. Gamage, C. Xu, R. R. Kompella, and D. Xu, "vPipe: Piped I/O Offloading for Efficient Data Movement in Virtualized Clouds," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SOCC 2014)*. New York, NY, USA: ACM, 2014, pp. 27:1–27:13.

[13] K. Dey, D. Mishra, and P. Kulkarni, "Vagabond: Dynamic Network Endpoint Reconfiguration in Virtualized Environments," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SOCC 2014)*. New York, NY, USA: ACM, pp. 21:1–21:13.

[14] H. Kim, S. Kim, J. Jeong, and J. Lee, "Virtual Asymmetric Multiprocessor for Interactive Performance of Consolidated Desktops," in *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2014)*. New York, NY, USA: ACM, 2014, pp. 29–40.

[15] D. Li, H. Jin, X. Liao, and J. Yu, "Improving Write Amplification in a Virtualized and Multimedia SSD System," *Multimedia Tools and Applications*, pp. 1–21, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11042-013-1497-6

[16] "Amazon Online Shopping," 2015, http://www.amazon.com.

[17] K. Razavi and T. Kielmann, "Scalable Virtual Machine Deployment Using VM Image Caches," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2013)*. New York, NY, USA: ACM, 2013, pp. 65:1–65:12.

[18] F. Chen, T. Luo, and X. Zhang, "CAFTL: A Content-aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST 2011)*. Berkeley, CA, USA: USENIX Association, 2011, pp. 77–90.

[19] K. Shen, S. Park, and M. Zhu, "Journaling of Journal is (Almost) Free," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 2014)*. Berkeley, CA, USA: USENIX Association, 2014, pp. 287–293.

[20] A. Ma, C. Dragga, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "ffsck: The Fast File System Checker," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*. San Jose, CA: USENIX, 2013, pp. 1–15.

[21] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, and Y. Bao, "CMD: Classification-based Memory Deduplication Through Page Access Characteristics," in *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2014)*. New York, NY, USA: ACM, 2014, pp. 65–76.

[22] R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala, and M. Zhao, "Write Policies for Host-side Flash Caches," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 2013)*. San Jose, CA: USENIX, 2013, pp. 45–58.

[23] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "COMMA: Coordinating the Migration of Multi-tier Applications," in *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2014)*. New York, NY, USA: ACM, 2014, pp. 153–164.

[24] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, "XvMotion: Unified Virtual Machine Migration over Long Distance," in *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 2014)*, Jun. 2014, pp. 97–108.

[25] "LZ4: Extremely Fast Compression Algorithm," 2014, https://code.google.com/p/lz4/.

[26] D. Harnik, R. Kat, D. Sotnikov, A. Traeger, and O. Margalit, "To Zip or Not to Zip: Effective Resource Usage for Real-Time Compression," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 2013)*. San Jose, CA: USENIX, 2013, pp. 229–241.

[27] Y. Xiang, H. Liu, T. Lan, H. Huang, and S. Subramaniam, "Optimizing Job Reliability via Contention-free, Distributed Scheduling of VM Checkpointing," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC 2014)*. New York, NY, USA: ACM, 2014, pp. 59–64.

[28] D. Le, H. Huang, and H. Wang, "Understanding Performance Implications of Nested File Systems in a Virtualized Environment," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST 2012)*. San Jose, CA: USENIX, 2012.

[29] "Iometer benchmark," 2014, http://www.iometer.org.

[30] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo Benchmarks: Java Benchmarking Development and Analysis," in *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA 2006)*. New York, NY, USA: ACM, 2006, pp. 169–190.

[31] N. Zeldovich and R. Chandra, "Interactive Performance Measurement with VNCPlay," in *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*. USENIX, 2005, pp. 189–198.

[32] C. Daskalakis, I. Diakonikolas, and R. A. Servedio, "Learning Poisson Binomial Distributions," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC 2012)*. New York, NY, USA: ACM, 2012, pp. 709–728.

[33] Z. Chen, Y. Zhao, X. Miao, Y. Chen, and Q. Wang, "Rapid Provisioning of Cloud Infrastructure Leveraging Peer-to-Peer Networks," in *Proceedings of the 29th IEEE International Conference on ICDCS Workshops '09.*, June 2009, pp. 324–329.

[34] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers," in *Proceedings of the 2012 IEEE INFOCOM*, March 2012, pp. 181–189.

[35] B. Nicolae, J. Bresnahan, K. Keahey, and G. Antoniu, "Going Back and Forth: Efficient Multideployment and Multisnapshotting on Clouds," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC 2011)*. New York, NY, USA: ACM, 2011, pp. 147–158.

[36] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein, "VMTorrent: Scalable P2P Virtual Machine Streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. New York, NY, USA: ACM, 2012, pp. 289–300.

[37] M. Shamma, D. T. Meyer, J. Wires, M. Ivanova, N. C. Hutchinson, and A. Warfield, "Capo: Recapitulating Storage for Virtual Desktops," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST 2011)*. Berkeley, CA, USA: USENIX Association, 2011, pp. 31–45.

[38] C. Tang, "FVD: A High-Performance Virtual Machine Image Format for Cloud," in *Proceedings of the 2011 USENIX conference on USENIX Annual Technical Conference (USENIX ATC 2011)*. Berkeley, CA, USA: USENIX Association, 2011.

[39] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "IOFlow: A Software-defined Storage Architecture," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP 2013)*. New York, NY, USA: ACM, 2013, pp. 182–196.

[40] J.-P. Billaud and A. Gulati, "hClock: Hierarchical QoS for Packet Scheduling in a Hypervisor," in *Proceedings of the 8th ACM European Conference on Computer Systems (Eurosys 2013)*. New York, NY, USA: ACM, 2013, pp. 309–322.

[41] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "PriorityMeister: Tail Latency QoS for Shared Networked Storage," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SOCC 2014)*. New York, NY, USA: ACM, 2014, pp. 29:1–29:14.