HAL
open science

Towards Efficient On-demand VM Provisioning: Study of VM Runtime I/O Access Patterns to Shared Image Content

Andrzej Kochut, Alexei Karve, Bogdan Nicolae

To cite this version:

Andrzej Kochut, Alexei Karve, Bogdan Nicolae. Towards Efficient On-demand VM Provisioning: Study of VM Runtime I/O Access Patterns to Shared Image Content. IM'15: 13th IFIP/IEEE International Symposium on Integrated Network Management, May 2015, Ottawa, Canada. 10.1109/INM.2015.7140307 . hal-01138689

HAL Id: hal-01138689

https://inria.hal.science/hal-01138689v1

Submitted on 2 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards Efficient On-demand VM Provisioning: Study of VM Runtime I/O Access Patterns to Shared Image Content

Andrzej Kochut, Alexei Karve
IBM T.J. Watson Research Center
1101 Kitchawan Road, Route 134
Yorktown Heights, N.Y. 10598, USA
{akochut,karve}@us.ibm.com

Bogdan Nicolae
IBM Ireland Research Center
Damastown Industrial Park
Mulhuddart, Dublin 15, Ireland
bogdan.nicolae@ie.ibm.com

***Abstract*—IaaS clouds are becoming a standard way of providing elastic compute capacity at an affordable cost. To achieve that, VM provisioning system has to optimally allocate I/O and compute resources. One of the significant optimization opportunities is to leverage content similarity across VM images. While many studies have been devoted to de-duplication of VM images, this paper is to the best of our knowledge, the first one to comprehensively study the relationship between the VM image similarity structure and the runtime I/O access patterns. Our study focuses on block-level similarity and I/O access patterns, revealing correlations between common content of different images and application-level access semantics. Furthermore, it also zooms on several runtime I/O access pattern aspects, such as the similarity of the sequences in which common content is accessed. The results show a strong tendency for access pattern locality within common content clusters across VM images, regardless of the rest of the composition. Furthermore, it reveals a strong tendency for read accesses to refer to the same subsets of blocks within the common content clusters, while preserving the same ordering. These results provide important insights that can be used to optimize on-demand VM image content delivery under concurrency.**

I. Introduction

One of the main features that has contributed to the growing popularity of Infrastructure clouds (Infrastructure-as-a-Service, or IaaS clouds) is the elastic on-demand provisioning of resources: users can bring up a whole virtual cluster made out of virtual machine instances and reconfigure it dynamically with a simple click of a button [32], [3]. However, behind the simplicity exposed to the users lie difficult challenges. On one hand, there is a need to minimize the provisioning time and guarantee scalability despite a growing number of VMs, otherwise users do not perceive IaaS as truly on-demand and lose interest. At the same time, cloud providers need to store and manipulate the VM images used to provision the VM instances with minimal resource utilization (storage space, I/O bandwidth), otherwise losing potential profit by not allocating these resources to actual user workloads.

In this context, techniques to provision VM instances saw two major developments. One way is to *pre-copy* the entire image from the image repository to the local file system of the target hypervisor before starting up a VM instance. Since disk images can quickly grow in the order of tens of GBs, a pre-broadcast can take in the order of tens of minutes or even hours [35], not counting the time spent afterwards to actually boot and run the VM instance. Most of the time, this approach is sub-optimal because of two reasons: (1) not all content of the image is read; and (2) reads from the image need to wait for the whole pre-copy to finish. For this reason, on-demand techniques were developed that bring the content of the VM image as needed while the VM instance is running (e.g., locally derived copy-on-write images – such as QCOW2 – that use a remotely stored VM image template as a read-only backing file). However, despite fixing the issues of pre-copy, on-demand techniques are not without drawbacks: reading parts of the remote VM image is an expensive operation in terms of latency and I/O bandwidth consumption. Even if the performance overhead is not significant [5] for single instances, at large scale where a large number of VM instances need to share and access the remote VM images concurrently, there is a high degree of I/O contention to the repository where the images are stored.

Under such circumstances, the I/O bandwidth can be quickly exhausted, which slows down not only the provisioning of the VM instances itself, but all other I/O to the repository as well (e.g. reads/writes of user datasets). This problem is further exacerbated by the fact that VM image repositories often apply redundancy elimination techniques based on de-duplication to reduce the storage space and I/O bandwidth necessary to transfer the VM image content to VM instances. Several studies [13], [11] show a large degree of overlap of content between VM images, which can go as high as 94%. Thus, when storing only unique content on the repository, contention to the same pieces of data will rise even further.

In an attempt to alleviate the I/O pressure on the repository, peer-to-peer content delivery networks with the VM instances have been proposed [25]. Such approaches enable hypervisors to directly exchange the image content without involving the repository. Also, data redundancy across images have been exploited to optimize VM provisioning [16], which allows to reuse locally present content from another image instead of transferring it from the repository. In this context, the aspect of adapting to the read access pattern is critical. It is important to exchange only what data is needed and to do so as early as possible in order to mask I/O latency. However, despite extensive studies on how the content of VM images

overlaps and what de-duplication potential can be achieved, there is little understanding in terms of how identical content (potentially belonging to different VM instances) behaves in relationship to the actual read access pattern issued by the VM instances during runtime. Such understanding could provide critical hints for the design of efficient on-demand provisioning techniques based on content delivery networks by introducing new optimization opportunities based on de-duplication. Thus, it is important to reveal questions like: what pieces of unique content are accessed in the first place? Which unique content is accessed more frequently and can lead to contention? Are there any orderings or temporal dependencies observable for the unique pieces accessed by the VM instances even if the original accesses refer to different images and different offsets?

This paper contributes with a comprehensive study that aims to answer the questions mentioned above. To the best of our knowledge, we are the first to explore such issues. Specifically, the key contributions of this paper are:

- Analysis of the content overlap structure (by computing 512-byte block digests and identifying matching blocks across VM images) for a set of representative combinations of OS versions and applications. The study covers several combinations of popular operating systems and middleware packages. The static analysis is augmented by the characterization of I/O access volumes to each of the content clusters during a typical execution.
- Correlation of runtime I/O access patterns with content similarity clusters revealed during the similarity analysis. In particular, we aim to understand: (1) what blocks within a shared cluster are actually touched during runtime; (2) whether these blocks remain the same even when the images that share the cluster are different; and (3) whether their offsets within the images are the same.
- Analysis of temporal aspects exhibited by the runtime I/O access patterns in relationship to content similarity. In particular, we aim to understand whether different pairs of VM instances (based on different images) access identical blocks in a similar order. This aspect is important in the design of on-demand run-time optimization for the purpose of prioritizing prefetching of blocks.

The remainder of this paper is organized as follows: Section II discusses related work, Section III describes static content overlap analysis. Section IV focuses on correlation of runtime I/O access patterns with content similarity clusters, analyzes I/O accesses within shared content clusters, and also examines temporal access patterns. Finally, Sections V and VI summarize our findings and outline future research directions.

II. Related work

On-demand instantiation of virtual machines based on images stored at a remote repository was shown to incur little runtime overhead [5] and is already an industry-standard practice, thanks to image formats such as *qcow2*: the remote image is used as a read-only template, while all modifications are stored locally in a separate file. Using a centralized file server as a repository is still a popular choice [28]. Decentralized solutions see increasing adoption in form of parallel file systems [29] or dedicated repositories [24].

Content similarity detection is typically performed by means of de-duplication, which is broadly classified into *static* and *content-defined*. Static approaches split the input data into equally sized chunks, which are then compared among each other (either byte-by-byte or, for increased performance, based on their hash values) in order to identify and eliminate duplicates. While simple and fast, static approaches suffer from misalignment issues (i.e insertions or deletions lead to the impossibility to detect duplicates). To deal with such misalignment issues, *content defined* approaches [22] were proposed. Essentially, they involve a sliding window over the data and that hashes the window content at each step using Rabin's fingerprinting method [27]. Many storage systems have adopted and refined de-duplication techniques [37], [8], [9], [23].

Specifically in the context of IaaS clouds, de-duplication techniques have proven effective at reducing the amount of space and network bandwidth necessary to store and transfer VM images. Several studies report significant reductions that can reach up to 94% [13], [11]. This potential has been exploited in dedicated VM image repositories such as VMAR [30] in order to store VM images in a de-duplicated fashion and redirect accesses for identical blocks to the same location before going through the OS. Utilizing content similarity in workloads from production storage systems is discussed in [17]. Several optimizations specifically targeting the performance of reading de-duplicated data during on-demand accesses have also been recently proposed [19]. Furthermore, outside of storage, de-duplication has demonstrated important benefits in the area of live migration [6], [2].

Analysis of I/O access patterns has been performed in a variety of contexts and at various granularity. At block-level, several studies [18], [34], [20] aim to discover disk I/O bottlenecks and optimization opportunities related to caching, prefetching, data layout and disk scheduling. At file-level, areas of interest include: prefetching optimizations [36], placement strategies to reduce energy consumption [10], inter-file access patterns in the context of collective I/O [21], file-access patterns of data-intensive workflow applications [31]. Furthermore, many studies focus on analysis of distributed storage systems as a whole: file popularity, temporal locality and arrival patterns for HDFS [1], key-value stores [4], leadership class parallel file systems [15].

This contribution focuses a particular aspect: understanding correlations between the content similarity of VM images and the I/O access patterns generated during the lifetime of VM instances deployed from the VM images using on-demand techniques. To our best knowledge, *we are the first to explore such correlations*.

III. VM Image Composition

The first step in our exploration is to understand how the VM images used to launch VM instances overlap in terms of content. To this end, we use a static de-duplication approach that splits each image in fixed sized blocks of 512 bytes (several studies [7], [14], [12], [26] of VM image similarity

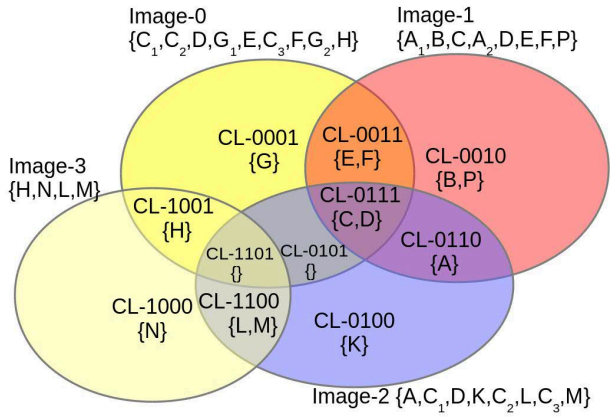


Fig. 1. Illustration of content clusters for four VM images. Upper case letters denote non-identical blocks. Identical blocks can appear more than once in the same image and are differentiated by the use of subscript

have confirmed the effectiveness of fixed block de-duplication in the context of VM images). Then, all blocks from all images are compared with each other (based on SHA-1 digests) and all duplicates are eliminated. The remaining unique blocks are grouped together into clusters, with each cluster comprising blocks that appear in a different subset of images and each block is represented by its hash value.

We briefly detail the concept of clusters. For simplicity, consider four partially overlapping images numbered Image-0 through Image-3 composed out of blocks labeled by upper case letters as shown in Fig. 1. Each upper-case letter denotes a globally unique block with respect to content, i.e. the same block that occurs in multiple VM images will carry the same upper-case letter. Since the same block can occur multiple times in the same image, each such multiple occurrence is uniquely identified by the use of subscript.

Each cluster comprises the maximal set of unique blocks that occurs at the intersection of a given combination of VM images without occurring in any other combination. It is labeled with a four-bit signature: the least significant refers to Image0 while the most significant refers to Image3. The value of the bit indicates whether the image is considered in the intersection or not.

There are four singleton clusters, each containing blocks unique to their respective images: CL-0001 has block G present only in Image-0, CL-0010 has blocks B and P present only in Image-1, CL-0100 has block K present only in Image-2 and finally CL-1000 has block N present only in Image-3. Block G in singleton cluster CL-0001 occurs two times in Image-0, as indicated by the use of subscript and represents internal redundancy within the same image where a the same block occurs at different offsets. The rest of the clusters are shared between images. For example, cluster CL-0011 has blocks with hash values E and F, both these blocks are shared by two images Image-0 and Image-1. The block with the hash value C is shared between E and F in Image 0. Also, Block C occurs at three different locations in Image-0 while it occurs only once in Image-1. Thus, blocks identified in the cluster may be non-contiguous in any image and may occur a different number of times in different images. Also, note that CL-1101 is empty, i.e. there are no blocks in common between Image-0, Image-2 and Image-3.

In order to achieve this clustering, we start from an initial cluster corresponding to the first VM image. Then, we gradually from new clusters by adding new images one-by-one. More specifically, when a new image is added, we compute for each already existing clusters the intersection with the the new image to be added (in terms of unique SHA-1 fingerprints corresponding to blocks). Next, the label of the cluster is extended by one bit. If the intersection is empty, then the cluster is unchanged except for the additional bit in the label, which is set to 0. Otherwise, the cluster is split into two new clusters: one corresponding to the intersection (additional bit set to 1) and the other one corresponding to the content not present in the new image (additional bit set to 0). If any fingerprints are left over after checking against all existing clusters, an additional cluster corresponding to the unique content of the new image is created to hold them (additional bit set to 1, the rest set to 0). We use several optimizations to accelerate the cluster construction, including bloom filters and multi-threaded parallelism. A more detailed description of this process and its optimizations can be found in our previous work [14].

Using this approach, we analyze several variations of VM images that include two versions of the GNU/Linux operating system (Fedora Core 19 and 20), IBM DB2 version 10, two versions of Apache Tomcat application container (versions 7.0 and 8.0), as well as two versions of MySQL database engine (versions 5.1 and 5.5). From the numerous variations and combinations possible, these are relevant and representative of frequent VM image content (i.e. OS, web, database stack is very popular) and common access patterns (where an OS always needs to boot and a database, if needed, always needs to start). In practice, users upload a large number of VM images and spawn multiple instances from them, yet there is a limited number of popular choices for OS, DB engines, web stacks, etc. that are typically used to "cook" VM images. This introduces a high chance for similarity, which we argue offers a large potential for runtime optimization.

Table I summarizes the observed overlap structure, where only the non-empty clusters larger than 5 MB (a total of 24) are included. Each row corresponds to a cluster and each column corresponds to a VM image with a specific software stack. The values are sizes of the content (in MB) contained within a cluster. For example, cluster *cl-05* is common for all VM images in the study and corresponds to the content present in both Fedora Core 19 and 20 (and thus representative of any image based on these operating systems). Cluster *cl-11* contains content specific to MySQL version 5.5 but shared across both Fedora Core (FC) 19 and 20. On the contrary, cluster *cl-10* contains content specific MySQL version 5.5 on Fedora Core 19. A comparative illustration of the similarity clusters grouped by the installed software stack is depicted in Figure 2.

The analysis of the overlap structure leads to an important observation that clusters formed by content analysis at block device level (which does not have any semantic information available at file system level within the VM guest) reflect very well the application content installed within VM images. The analysis allows to distill content fragments and relate it to software packages. For example, for MySQL there are two

TABLE I. CONTENT OVERLAP STRUCTURE ACROSS 12 VIRTUAL MACHINE IMAGES USED IN THE ANALYSIS. COLUMNS CORRESPOND TO SPECIFIC SOFTWARE STACKS WHILE ROWS TO CLUSTERS SHARED ACROSS THE STACKS. SIZES IN MB.

Cluster	Fedora Core 19						Fedora Core 20					
	OS	DB2	MySQL		Tomcat		OS	DB2	MySQL		Tomcat	
	only	10	5.1	5.5	7.0	8.0	only	10	5.1	5.5	7.0	8.0
cl-01	0	21	0	0	0	0	0	0	0	0	0	0
cl-02	0	1952	0	0	0	0	0	1952	0	0	0	0
cl-03	27	27	0	0	27	27	0	0	0	0	0	0
cl-04	2483	2483	2483	2483	2483	2483	0	0	0	0	0	0
cl-05	825	825	825	825	825	825	825	825	825	825	825	825
cl-06	0	0	17	0	0	0	0	0	0	0	0	0
cl-07	0	0	298	0	0	0	0	0	298	0	0	0
cl-08	0	0	153	153	0	0	0	0	0	0	0	0
cl-09	0	0	78	78	0	0	0	0	78	78	0	0
cl-10	0	0	0	18	0	0	0	0	0	0	0	0
cl-11	0	0	0	415	0	0	0	0	0	415	0	0
cl-12	0	0	0	0	3	0	0	0	0	0	0	0
cl-13	0	0	0	0	8	0	0	0	0	0	8	0
cl-14	0	0	0	0	4	4	0	0	0	0	4	4
cl-15	0	0	0	0	0	5	0	0	0	0	0	0
cl-16	0	0	0	0	0	8	0	0	0	0	0	8
cl-17	0	0	0	0	0	0	0	153	0	0	0	0
cl-18	0	0	0	0	0	0	13	13	0	0	13	13
cl-19	0	0	0	0	0	0	2529	2529	2529	2529	2529	2529
cl-20	0	0	0	0	0	0	0	0	21	0	0	0
cl-21	0	0	0	0	0	0	0	0	153	153	0	0
cl-22	0	0	0	0	0	0	0	0	0	15	0	0
cl-23	0	0	0	0	0	0	0	0	0	0	4	0
cl-24	0	0	0	0	0	0	0	0	0	0	0	9
Total	3337	5311	3857	3975	3353	3354	3368	5474	3906	4018	3385	3390

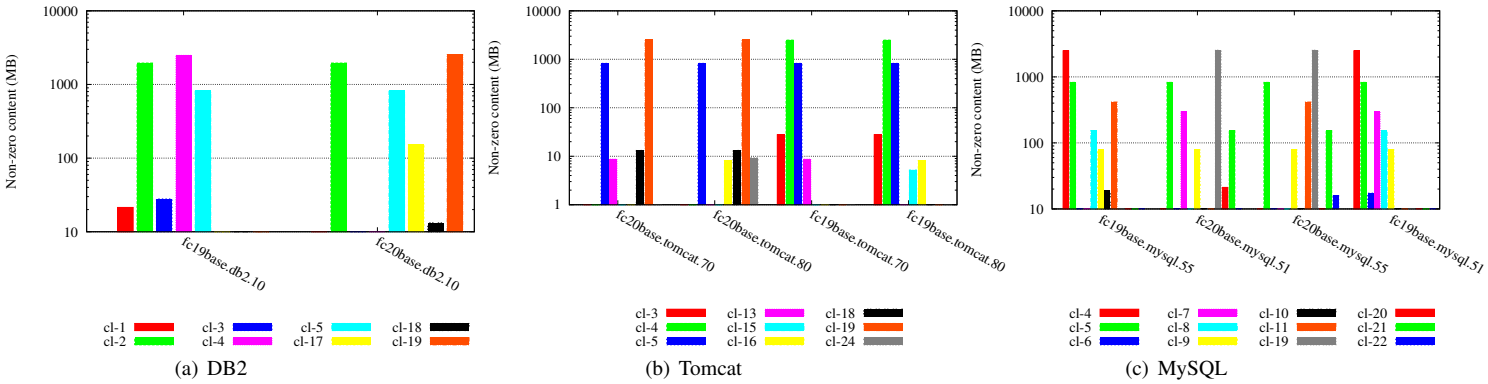


Fig. 2. Comparative illustration of the similarity clusters detailed in Table I grouped by installed software stack. Note the logscale on the X axis.

versions of software installed on two versions of the operating system. Content related to these installations falls into four categories. The first is content unique to 5.1 on FC 19 (17MB in cluster *cl-06*), unique to 5.5 on FC 19 (18MB in cluster *cl-10*, unique to 5.1 on FC 20 (21MB in cluster *cl-20*), unique to 5.5 on FC 20 (18MB in cluster *cl-22*. Next category is content shared between versions 5.1 and 5.5 within OS releases (153MB in *cl-08* and in *cl-21* for FC 19 and 20, respectively) and content shared both across the versions of MySQL and OS (78MB in *cl-09*). Finally, there is content unique to specific MySQL version, but common across operating system versions (298MB in *cl-07* for 5.1 and 415MB in *cl-11* for 5.5). Similar situation occurs for Tomcat 7.0 and 8.0 as well as DB2. It shows that clustering links well to software installed within the OS. Moreover, there is a substantial amount of content shared across all VM images with the same OS (cluster *cl-04* for Fedora Core 19 and *cl-19* for Fedora Core 20) and also across different OS releases (*cl-05*). The latter is due to the code base that has not been updated between the releases.

Another interesting observation is the filesystem's behavior while installing MySQL packages. Observe that the cluster *cl-03* is shared among all images based on FC 19 except for the ones with MySQL installed. That is due to the fact that MySQL installation is done using yum package installer

which, in addition to installing MySQL rpms, also downloads and upgrades several existing packages. As a result, it modifies data (17MB in that case) which becomes shared by only images with no MySQL.

IV. VM RUNTIME ACCESS PATTERNS

This section analyses the read access patterns generated during the runtime of VM instances in the context of the image content overlap structure described in Section III.

A. Experimental Configuration

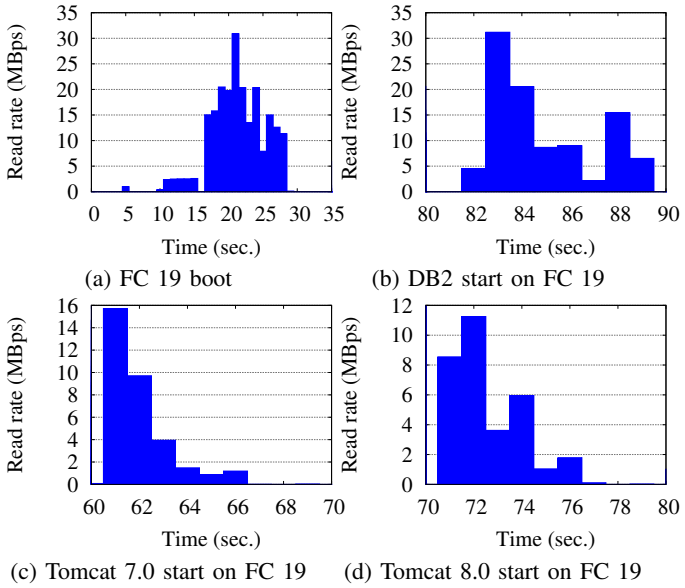
In order to capture access patterns of the VM instances, we implemented a FUSE-based [33] filesystem that traces all read operations in terms of access time, the offset within the image and the number of blocks requested. To avoid the complexity of dealing with write operations and modifications to the image, all VM instances use derived *qcow2* copy-on-write images as virtual disks, which in turn are based on the original images that are stored in the FUSE filesystem in the raw format.

The experiments were conducted on physical servers leased from SoftLayer [32] and involve booting a VM instance as described above, then optionally launching a specific workload while capturing the read accesses. We use the various configurations described in Section III as the base images. With respect to the workloads, we have chosen three representative use cases: (1) launching DB2, representative of large applications that occupy a large amount of space and exhibit a large initialization overhead; (2) launching Tomcat and going through the typical deployment cycle of a servlet (e.g., launch admin UI, browse documentation, issue requests, etc.), which is representative of web service management; (3) launching OS tools and utilities (e.g. grep, awk, find) in various combinations, which is representative of scripting workloads (e.g. log analysis). After obtaining the traces for various combinations of images and workloads, we analyze these traces both individually and in selected pairs in order to identify correlations.

B. Runtime Read Rates

Virtual disk access patterns of VM instances are driven primarily by the workload executing in the guest. Usually, a relatively small set of applications from the installed content is actually launched and executed. Similarly, even when an application is launched, the read footprint may change significantly based on invoked functionality.

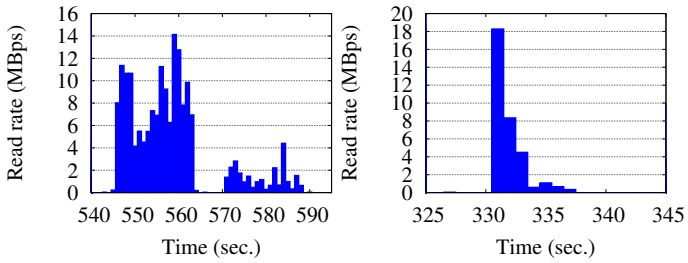
When observing the evolution of the read throughput during runtime, we see that on our test system it reaches up to 30MBps for well tuned VM instances. Figure 3 presents sample results for four scenarios. Figure 3a shows boot sequence for Fedora Core 19. It lasts under 30 seconds and generates between 2 and 30 MBps reads against the base disk (when averaged over 1 second intervals). The launch of applications is much shorter with the results for DB2 (Figure 3b) and two versions of Tomcat (Figures 3c and 3d) indicating up to 6 second bursts and a similar range for the throughput. Figure 4a shows the read rate (averaged per second) during an execution representative of scripting workloads and Figure 4b shows activity for interaction with Tomcat 7.0 on FC 20, representative of web service management. Workload specifics are described in Section IV-A.



(a) FC 19 boot (b) DB2 start on FC 19

(c) Tomcat 7.0 start on FC 19 (d) Tomcat 8.0 start on FC 19

Fig. 3. Read rates for launch activities on FC 19.



(a) FC 20 and scripting workload (b) Tomcat 7.0 launch and web service management

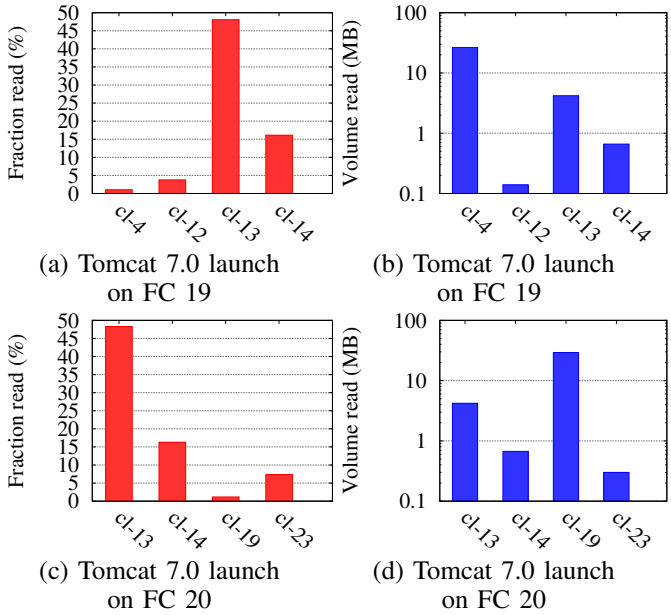
Fig. 4. Read rates for activities post launch.

C. Semantic Locality to Cluster Boundaries

Next, we analyze the correlation between the read activities and content sharing structure (clusters) identified in Section III. We want to find out whether the activity for an application results in reads to the cluster related only to this application or also to other clusters. To do so, each 512-byte read operation captured at runtime was matched with the cluster that it belongs to.

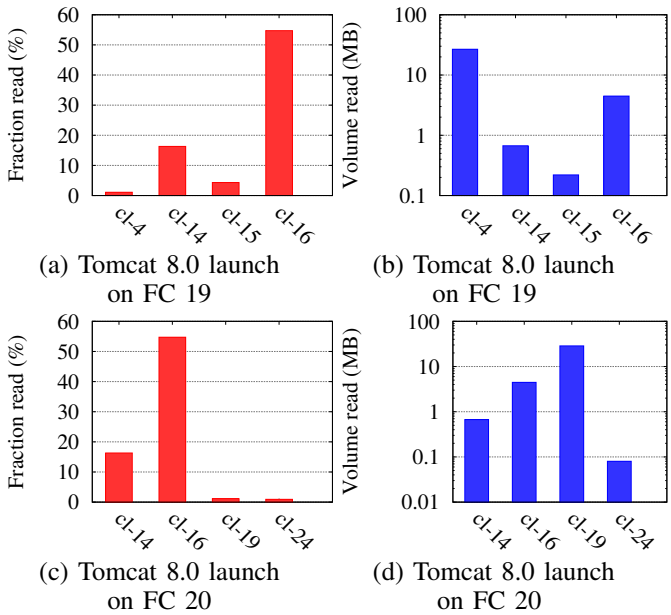
Figure 5 presents results for launching Tomcat 7.0 on FC 19 and FC 20. Figures 5a and 5c depict fractions of clusters that were read during the launch of Tomcat on FC 19 (a) and FC 20 (b). Figures 5b and 5d present related volumes of reads (in MB). The clusters accessed by starting Tomcat 7.0 are *cl-4, cl-12 cl-13* and *cl-14*. Referring back to static overlap structure shown in Table I we see that *cl-12* contains content unique to Tomcat 7.0 on FC 19, *cl-13* contains content unique to Tomcat 7.0 independent of OS version, and *cl-14* contains content shared between Tomcat 7.0 and 8.0 also independent of OS version. Finally, *cl-4* contains content common to FC 19 (OS blocks). Almost half of content in the shared cluster *cl-13* is used. Overall volumes of reads are relatively small since the Tomcat installation footprint is small.

Similar situation can be observed in case of Tomcat 8.0 starting on FC 19 (Figure 6). There are two common Tomcat clusters, *cl-14* and *cl-16*, related to content common across



(a) Tomcat 7.0 launch on FC 19
(b) Tomcat 7.0 launch on FC 19
(c) Tomcat 7.0 launch on FC 20
(d) Tomcat 7.0 launch on FC 20

Fig. 5. Fraction and volumes of base image reads while starting Tomcat 7.0 on FC 19 and 20.



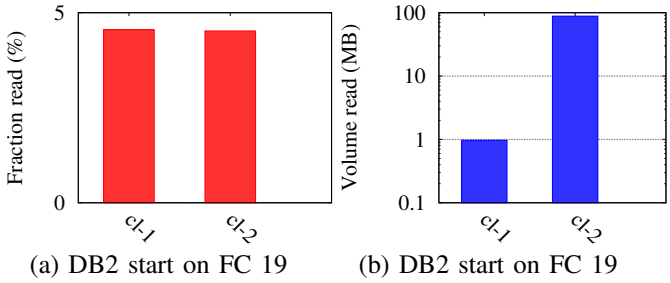
(a) Tomcat 8.0 launch on FC 19
(b) Tomcat 8.0 launch on FC 19
(c) Tomcat 8.0 launch on FC 20
(d) Tomcat 8.0 launch on FC 20

Fig. 6. Fraction and volumes of base image reads while starting Tomcat 8.0 on FC 19 and 20.

both Tomcat versions and common to Tomcat 8.0, respectively. Note that *cl-14* was also used during launch of Tomcat 7.0 given it is shared between the two. Clusters *cl-15* and *cl-24* are unique to Tomcat 8.0 on FC 19 and 20, respectively. Clusters *cl-4* and *cl-19* are related to OS content for FC 19 and 20, respectively.

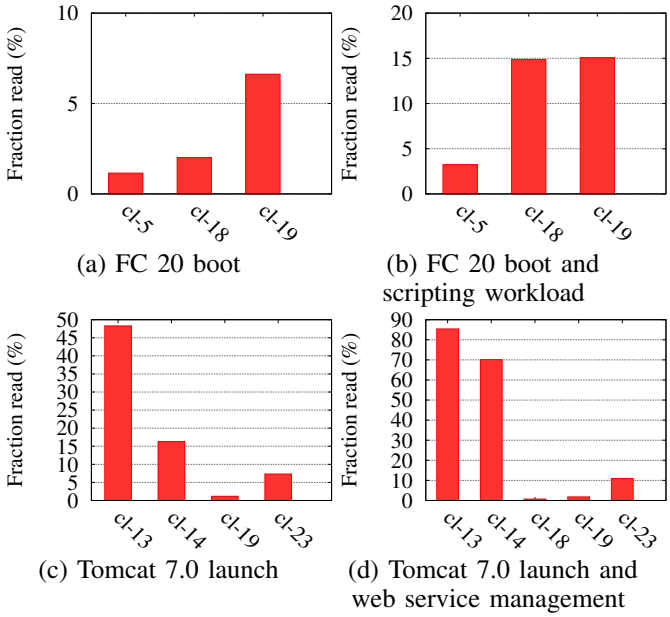
Based on the two cases above, it is worth noting that the accesses target both content specific to application being used as well as additional content from OS that is loaded as a result of usage of system libraries.

Next, let's consider start of IBM DB2 version 10 on FC 19. This scenario is depicted in Figure 7 that shows both fractions



(a) DB2 start on FC 19
(b) DB2 start on FC 19

Fig. 7. Fraction and volumes of base image reads while starting DB2 10 on FC 19.



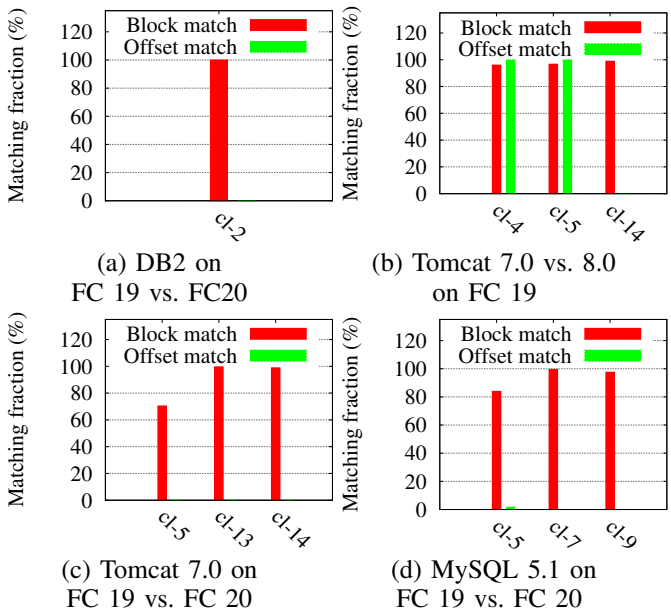
(a) FC 20 boot
(b) FC 20 boot and scripting workload
(c) Tomcat 7.0 launch
(d) Tomcat 7.0 launch and web service management

Fig. 8. Comparison of fraction of clusters read from base images during launch and activities.

of clusters read in (a) and volumes read in (b). Only two clusters are being used, *cl-1* and *cl-2*, both directly related to DB2. Note that *cl-2* is shared across both FC 19 and FC 20. Also, only small fraction of DB2 content is used during launch (just under 5%), however it results in more than 100MB of reads.

Finally, let's consider the effects of using an OS or application in steady-state, i.e., using more OS programs or application features. This situation for FC 20 and Tomcat 7.0 is depicted in Figure 8.

Figure 8a shows content read during OS boot itself while 8b shows activity during launching OS tools and utilities (as described in Section IV-A). First, all reads were issued to the same clusters as during the boot, confirming semantic locality. Moreover, as expected, the fractions of clusters accessed has increased significantly. For *cl-19* (which contains majority of FC 20 OS content) it more than doubled from 6.5% to 15%. For *cl-18* the increase was even larger from 2% to 15%. Both clusters are shared across FC 20 instances (with exception of MySQL that was explained in Section III). Similar situation is observed for launching Tomcat and going through the typical deployment cycle of a servlet (e.g., launch admin UI, browse documentation, issue requests, etc.). Usage of shared clusters *cl-13* and *cl-14* grew significantly. The usage in case of *cl-13*



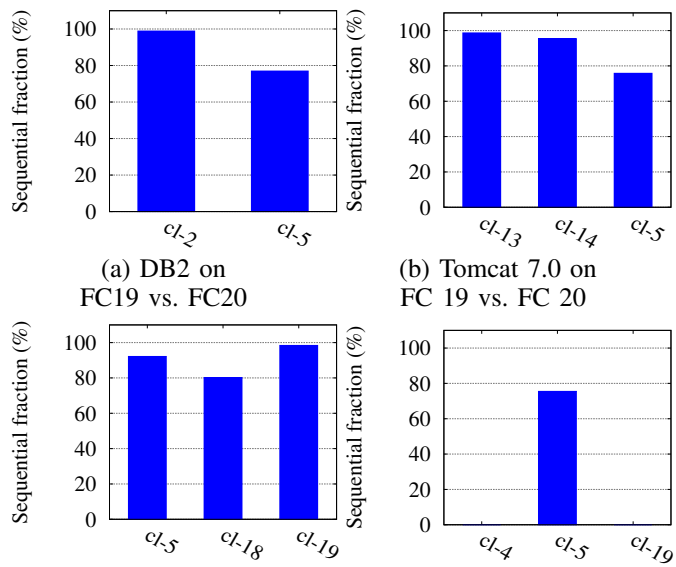
(a) DB2 on FC 19 vs. FC20

(b) Tomcat 7.0 vs. 8.0 on FC 19

(c) Tomcat 7.0 on FC 19 vs. FC 20

(d) MySQL 5.1 on FC 19 vs. FC 20

Fig. 9. Per-cluster fractional match across pairs of VM instances based on different images.



(a) DB2 on FC19 vs. FC20

(b) Tomcat 7.0 on FC 19 vs. FC 20

(c) Two independent FC 20 boots

(d) Boot of FC 19 vs. FC 20

Fig. 10. Sequentiality match: fraction of common blocks accessed during the runtime of two VM instances that share the same predecessor.

rose from 47% to almost 90% and in case of *cl-14* rose from 15% to 70%. Also usage of cluster unique to Tomcat 7.0 on FC 20 (*cl-23*) rose from 6% to 10%.

D. Runtime Access Similarity

As shown in the previous section, the runtime read accesses are well localized within cluster boundaries that directly correspond to the semantic composition of the images (i.e. shared OS and application packages). However, one important aspect that is not revealed by the previous study is whether the accesses issued by different VM instances ultimately target the same blocks within a common cluster. Furthermore, when the instances are based on different images, the relative offsets of identical blocks of a common cluster may be different. Thus, it is important to understand not only how many common blocks were accessed from the same cluster, but also to what degree their offsets relative to the base image coincide, which ultimately reveals how the ordering of clusters inside an image reflects on the access pattern.

Figure 9 examines both aspects. Each plot corresponds to one pair of runtime access traces for two VM instances that are based on two different images that share common clusters. Represented is the fraction of common blocks and common offsets accessed by both instances during runtime, broken down by cluster.

Figure 9a shows the results for launching DB2 on FC 19 FC 20 respectively. Cluster *cl-2* captures most accesses (as show in Figure 7b). Furthermore, Figure 9a shows that almost 100% of the content read from this cluster is actually common between starting DB2 on FC 19 and 20. That confirms that in case of DB2 the accesses are not only localized to clusters, but also match the blocks almost exactly. Note, however, that most offsets in the image are shifted (only under 5% match), which stems from the fact that DB2 binaries ended up in different sections of virtual disk even though they were content-wise the same. This can be explained by the fact that the file system layout is affected by various factors, including previously installed and removed content, filesystem type, block allocation strategy, installed OS, etc.

Figure 9b shows comparison of launching Tomcat 7.0 and 8.0, both on FC 19. It shows that accesses on all three clusters match in terms of which blocks within a cluster are read (i.e., more than 90% matches). However, the offsets of the accessed blocks only matches for OS related clusters (*cl-4* and *cl-5* because both use the same base FC 19 OS image) and not for the Tomcat shared content cluster, since it is in different disk sections. Figures 9c and 9d show corresponding results for Tomcat 7.0 and MySQL 5.1 started on both FC 19 and 20, respectively. The shared clusters again match almost exactly for content, but the offset does not. For Tomcat, the fraction of common access between launching on FC 19 and 20 is only 70%, with the rest being unique to each launch. Similarly, the number for MySQL 5.1 is 82%.

E. Runtime Access Order within Clusters

The last aspect we study in our work is the degree of similarity between the access order of common blocks (as identified in prior sections) for the VM instances that share clusters. Such correlations are crucial in understanding how the temporal locality of read accesses of a VM instance relate to other instances.

In order to facilitate this understanding, we introduce a metric we call *sequentiality match*. Specifically, given two read traces of two VM instances *TR-1* and *TR-2* for a given cluster *cl*, we examine all 512-byte blocks from *cl* that are read in both *TR-1* and *TR-2* and keep only those blocks that share the same predecessor in both traces. The sequentiality match itself is the amount of the remaining blocks expressed as a fraction of the total number of common blocks accessed during the runtime.

Figure 10 shows the sequentiality match for several experiments. Figure 10a examines the case of DB2 starting on two different operating system versions. For cluster *cl-2* that contains DB2 specific content shared across OSes, the content is accessed in almost identical order, with sequentiality match measure at 98%. The second cluster has a lower but still significant value of 78%. The conclusion is similar in case of Tomcat on two OS versions, which is depicted in Figure 10b. The clusters specific to Tomcat (*cl-13* and *cl-14*) are accessed in almost identical order (metric value of above 95%), and the shared OS cluster *cl-5* has a very high sequentiality match that amounts to 78%. Figure 10c shows results for two separate boots of the same FC 20 OS, which captures potential variations during separate boots. A similar trend is observable for the common cluster *cl-5* in the case when FC 19 and FC 20 boot (Figure 10d), with the sequentiality match reaching close to 80%.

V. Summary of the Key Findings

In summary, we have confirmed that there is a significant overlap across VM images due to commonality of OS and application base. That leads to forming of an interesting sharing structure, exemplified in Table I, which reflects potential for runtime sharing. The fact that large fraction of content is shared across images indicates significant optimization potential for VM instance runtime.

Moreover, the study shows a very strong relation between runtime read access patterns and the content commonality structure based on clusters. More specifically, the blocks accessed during the runtime of the VM instances for a given workload tend to stay within the boundaries of those common clusters that comprise the OS and application packages related to the workloads. Also, when two VM instances access a common block, the corresponding offsets in the base images generally do not coincide, with exceptions observable only in the case when the base images are both derived from a common image, which tends to preserve the offsets in the clustering itself. Furthermore, when two VM instances read from the same cluster, they generally tend to access the same blocks, with the proportion of common blocks read from the same cluster varying between 70%-99%.

The amount of content actually accessed in each cluster is workload specific: it relates to how much functionality of a particular application or set of applications that fall inside the same cluster is actually used. As expected, with increased utilization of the same OS or application, more reads are expected to the same clusters. At the same time, cross-correlations between different clusters are observable as well: reads of blocks from some clusters trigger reads of blocks in other clusters, which can happen for example when certain applications invoke system packages (e.g. a Java application starts the Java Virtual Machine).

Finally, we observe that accesses tend to be sequenced in the same order independent of the software stack combination, with very high similarity of sequentiality observable (over 98%). Exceptions are mostly for OS content, where higher levels of access parallelism (e.g. daemons and tools launched in parallel) introduce more randomness. However, even for this situation the ordering of accesses is still quite similar, reaching 80%.

VI. Conclusions and Future Work

With a growing need to deploy more VM instances based on increasingly larger virtual disk images, the problem of efficient provisioning becomes a key challenge in the context of IaaS clouds. Two important advances were made in this regard. First, due to the high content similarity exhibited by VM images, storage repositories have successfully employed de-duplication techniques in order to reduce the space necessary to store them. Second, based on the observation that VM instances access only a small fraction of the virtual image content during runtime, on-demand provisioning techniques that access the repository only when needed have seen increasingly wider adoption compared with pre-copy techniques.

Despite important progress in both directions, most advances are targeting either one direction or another, with little effort and understanding dedicated to the problem of how to co-optimize de-duplication techniques and on-demand provisioning techniques. As a first step in this direction, this study has shown that there is a strong link between the on-demand read access patterns issued by VM instances during runtime and the block-level content similarity identified at virtual disk level.

More specifically, most reads that involve a particular application are concentrated inside the same block-level content similarity clusters, regardless of the rest of the composition of the virtual disk. This cluster locality extends throughout the usage of the application, including launch and steady-state use of various functionalities. Furthermore, the read accesses refer to the same subsets of blocks within the clusters, while preserving the same ordering. Again, these properties hold regardless of the rest of the composition of the virtual disk images, including various combinations of OS and software stacks.

Based on these findings, we plan to explore in future work how to design novel on-demand provisioning techniques that leverage content and access pattern similarity in order to reduce the performance overhead and network traffic involved in the provisioning of a large number of VM instances at scale under concurrency. We envision peer-to-peer content delivery mechanisms that automatically adapt to the access pattern and minimize any dependency on the storage repository while emphasizing direct content exchange in a decentralized and scalable fashion, effectively overcoming I/O bottlenecks created by contention to the repository.

References

[1] Cristina L. Abad, Nathan Roberts, Yi Lu, and Roy H. Campbell. A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns. In *IISWC '12 Proceedings of the 2012 IEEE International Symposium on Workload Characterization*, pages 100–109, San Diego, USA, 2012.

[2] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 159–170, San Jose, USA, 2011. ACM.

[3] Amazon Inc. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/, 2014.

[4] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *SIGMETRICS '12: Proceedings of the 12th ACM Joint International Conference on Measurement and Modeling of Computer Systems*, pages 53–64, London, UK, 2012. ACM.

[5] Han Chen, Minkyong Kim, Zhe Zhang, and Hui Lei. Empirical study of application runtime performance using on-demand streaming virtual disks in the cloud. In *MIDDLEWARE '12: Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, pages 5:1–5:6, Montreal, Quebec, Canada, 2012. ACM.

[6] Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. Live gang migration of virtual machines. In *HPDC '11: Proceedings of the 20th International Symposium on High Performance Distributed Computing*, pages 135–146, San Jose, USA, 2011. ACM.

[7] Fred Douglis, Jason Lavoie, John M. Tracey, Purushottam Kulkarni, and Purushottam Kulkarni. Redundancy elimination within large collections of files. In *In USENIX Annual Technical Conference, General Track*, pages 59–72, 2004.

[8] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: a scalable secondary storage. In *FAST '09: Proccedings of the 7th conference on File and storage technologies*, pages 197–210, San Francisco, USA, 2009. USENIX Association.

[9] Fanglu Guo and Petros Efstathopoulos. Building a high-performance deduplication system. In *USENIXATC'11: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, pages 25–39, Portland, USA, 2011. USENIX Association.

[10] Masaru Iritani and Haruo Yokota. Effects on performance and energy reduction by file relocation based on file-access correlations. In *EDBT-ICDT '12: Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 79–86, Berlin, Germany, 2012. ACM.

[11] K. R. Jayaram, Chunyi Peng, Zhe Zhang, Minkyong Kim, Han Chen, and Hui Lei. An empirical analysis of similarity in virtual machine images. In *Middleware '11: Proceedings of the Middleware 2011 Industry Track Workshop*, pages 6:1–6:6, Lisbon, Portugal, 2011. ACM.

[12] K. R. Jayaram, Chunyi Peng, Zhe Zhang, Minkyong Kim, Han Chen, and Hui Lei. An empirical analysis of similarity in virtual machine images. *Middleware*, 2011.

[13] Keren Jin and Ethan L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 7:1–7:12, Haifa, Israel, 2009. ACM.

[14] Alexei Karve and Andrzej Kochut. Redundancy aware virtual disk mobility for cloud computing. In *IEEE CLOUD*, pages 35–42, 2013.

[15] Youngjae Kim, R. Gunasekaran, G.M. Shipman, D.A Dillow, Zhe Zhang, and B.W. Settlemyer. Workload characterization of a leadership class storage cluster. In *PDSW '10: The 5th Petascale Data Storage Workshop*, pages 1–5, New Orleans, USA, Nov 2010.

[16] Andrzej Kochut and Alexei Karve. Leveraging local image redundancy for efficient virtual machine provisioning. *IEEE Network Operations and Management Symposium*, 2012.

[17] Ricardo Koller and RangaswamiShen Raju. I/o deduplication: Utilizing content similarity to improve i/o performance. In *FAST '10: Proceedings of the USENIX File and Storage Technologies*, pages 211–224. USENIX Association, 2010.

[18] Zhenmin Li, Zhifeng Chen, and Yuanyuan Zhou. Mining block correlations to improve storage performance. *Trans. Storage*, 1(2):213–245, May 2005.

[19] Bo Mao, Hong Jiang, Suzhen Wu, Yinjin Fu, and Lei Tian. Read-performance optimization for deduplication-based storage systems in the cloud. *Trans. Storage*, 10(2):6:1–6:22, March 2014.

[20] Manolis Marazakis, Vassilis Papaefstathiou, and Angelos Bilas. Optimization and bottleneck analysis of network block i/o in commodity storage systems. In *ICS '07: Proceedings of the 21st Annual International Conference on Supercomputing*, pages 33–42, Seattle, USA, 2007. ACM.

[21] Gokhan Memik, Mahmut T. Kandemir, Wei-Keng Liao, and Alok Choudhary. Multicollective i/o: A technique for exploiting inter-file access patterns. *Trans. Storage*, 2(3):349–369, August 2006.

[22] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. *SIGOPS Oper. Syst. Rev.*, 35(5):174–187, October 2001.

[23] Bogdan Nicolae. Towards Scalable Checkpoint Restart: A Collective Inline Memory Contents Deduplication Proposal. In *IPDPS '13: The 27th IEEE International Parallel and Distributed Processing Symposium*, pages 19–28, Boston, USA, 2013.

[24] Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. Going back and forth: Efficient multi-deployment and multi-snapshotting on clouds. In *HPDC '11: 20th International ACM Symposium on High-Performance Parallel and Distributed Computing*, pages 147–158, San José, USA, 2011.

[25] Bogdan Nicolae and Mustafa Rafique. Leveraging Collaborative Content Exchange for On-Demand VM Multi-Deployments in IaaS Clouds. In *Euro-Par '13: 19th International Euro-Par Conference on Parallel Processing*, pages 305–316, Aachen, Germany, 2013.

[26] Chunyi Peng, Minkyong Kim, Zhe Zhang, and Hui Lei. Vdn: Virtual machine image distribution network for cloud data centers. *IEEE NOMS*, 2012.

[27] Michael Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[28] Nicholas Aaron Robison and Thomas J. Hacker. Comparison of vm deployment methods for hpc education. In *RIIT '12: Proceedings of the 1st Annual conference on Research in Information Technology*, pages 43–48, Calgary, Canada, 2012. ACM.

[29] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA, 2002. USENIX Association.

[30] Zhiming Shen, Zhe Zhang, Andrzej Kochut, Alexei Karve, Han Chen, Minkyong Kim, Hui Lei, and Nicholas Fuller. Vmar: Optimizing i/o performance and resource utilization in the cloud. In *Middleware '13: Proceedings of the 14th ACM/IFIP/USENIX International Middleware Conference*, volume 8275, pages 183–203. Springer Berlin Heidelberg, 2013.

[31] Takeshi Shibata, SungJun Choi, and Kenjiro Taura. File-access patterns of data-intensive workflow applications and their implications to distributed filesystems. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 746–755, Chicago, USA, 2010. ACM.

[32] SoftLayer Company. SoftLayer Bare Metal Servers. http://www.softlayer.com/, 2014.

[33] SourceForge. FUSE Filesystem. http://fuse.sourceforge.net/.

[34] Xiaoyu Wang and Mitch Cherniack. Improving query i/o performance by permuting and refining block request sequences. In *CIKM '06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 652–661, Arlington, USA, 2006. ACM.

[35] Romain Wartel, Tony Cass, Belmiro Moreira, Ewan Roche, Manuel Guijarro, Sebastien Goasguen, and Ulrich Schwickerath. Image distribution mechanisms in large scale cloud providers. In *CloudCom '10: Proceedings of the 2nd IEEE Second International Conference on Cloud Computing Technology and Science*, pages 112–117, Indianapolis, USA, 2010. IEEE Computer Society.

[36] Peng Xia, Dan Feng, Hong Jiang, Lei Tian, and Fang Wang. FARMER: A novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 185–196, Boston, MA, USA, 2008.

[37] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 18:1–18:14, San Jose, USA, 2008. USENIX Association.