Towards Improved Provisioning and Utilization of Resources in Virtualized Environments

Synopsis

Submitted in partial fulfillment of the requirements
for the degree of

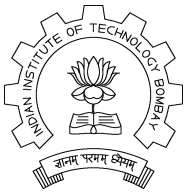
Doctor of Philosophy

by

Sujesha Sudevalayam
Roll No: 07305903

under the guidance of

Prof. Purushottam Kulkarni



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

1 Introduction and Motivation

With widespread adoption of virtualization for hosting applications, service providers (like Amazon EC2 [1]) can facilitate better performance isolation, security and elastic resource provisioning. A virtualization-based provisioning model is attractive for both providers [1]—multiplex resources among several customers, and clients—*pay-per-use*, use and pay for only as much resource as required. Instances of both *public* [1] and *private* [2] clouds exist, which leverage virtual machines for flexible provisioning.

Virtualization enables dynamic resource allocation and quicker deployment of services. Instead of installing new hardware for deploying a service, virtualization allows transparent, on-demand deployment on a few processors in an existing multi-processor machine and avoids new machine procurement delays. Virtualization allows accomodation of varying load-levels by on-demand resource allocation and also reduces application downtime. Due to above benefits of virtualization, many hosting centers have transitioned from providing Hardware as a Service (HaaS) to Infrastructure as a Service (IaaS) instead [1]. The primary difference between HaaS and IaaS is that the former involves use or leasing of physical hardware/machine whereas the latter involves leasing of virtual resources/machines.

Most web-based applications are multi-tiered and virtualization offers the possibility of hosting each of these tiers (e.g., the web-server tier, the application logic tier and the database server) on separate virtual machines. The major factors that affect the performance of a virtualized application are—available network capacity, disk access bandwidth and virtualization overheads. When multiple virtual machines (VMs) are placed on a single physical machine (PM), they compete for various resources like CPU, memory, network and disk I/O and interact in many conflicting ways. In any given virtualized environment, the physical resources available can be broadly categorized into the following:-

I. Resources allocated to the virtual machines. Every virtual machine has access to a set of resources, similar to those available on physical machines. For example, virtual CPU, VM page cache, virtual disk, etc.

II. Resources in the virtualized host. These are resources at the virtualized host to support and enable virtualization. For example, the hypervisor incurs CPU overheads for virtualization. The host's page cache is also used for buffering of physical I/O access, on behalf of VMs.

In this thesis, we address two important problems related to the management of both these types of resources more efficiently, towards the overall goal of optimizing the performance of virtualized applications and services.

Thesis contributions

1. Affinity-aware Modeling of CPU usage for Virtualized Applications. This problem deals with managing the network usage of VMs and estimating CPU requirement on both the VM and its host PM. Since different tiers of an application require mutual network communication, *colocation* of communicating VMs on the same PM reduces physical network usage. *Network affinity* is the presence of network communication between a pair of VMs, and is *intra-PM* when the VMs are colocated, and *inter-PM* when they are dispersed onto different PMs. Thus, the nature of network affinity is *mutable* (i.e., changing between inter-PM and intra-PM) upon VM migrations.

In our work, we perform network benchmarking, which demonstrates effects of network affinity on CPU usage when VMs are colocated versus dispersed. Next, we develop VM *pair-wise* models that can estimate the "colocated" CPU usage, on being input their individual dispersed-case resource usages. We also build similar models to estimate the "dispersed" CPU usage based on the individual colocated-case resource usages. For the "colocation" and "dispersion" models, we first built models that predicted the total (or absolute) CPU usage upon migration—these CPU models use all resource (CPU, disk, mutable and immutable network) usage profiles as their input. However, these models had an error of around 4%. So, next we built enhanced models to predict only the difference (or differential) CPU usage—these models use only the *mutable* network traffic metrics as input, and have maximum error within 2%. Finally, we demonstrated the application of *pair-wise* models to predict for multi-VM scenarios, with high accuracy.

2. Using Implicit Caching Hints for Disk I/O Reduction in Virtualized Environments. This problem deals with managing the cache usage on a virtualized host to improve disk access performance of VMs. Due to increased permeation of virtualization-based systems, there is a lot of inherent content similarity in systems like email servers, web servers and file servers. Harnessing content similarity can help avoid duplicate disk I/O requests that fetch the same content repeatedly. In this work, we incorporate intelligent I/O redirection within the storage virtualization engine of the device to manage the underlying block-cache like a content-cache.

We build a disk read-access optimization called DRIVE, that identifies content similarity across multiple blocks, and performs hint-based read I/O redirection. A metadata store is maintained and implicit caching hints are collected based on the VM's disk accesses. Using the hints, read I/O redirection is performed from within the VM's virtual block device, to manipulate the entire host-cache as a content-deduplicated cache. Our trace-based evaluation using a custom simulator, reveals that DRIVE achieves up to 20% better cache-hit ratios and reduces up to 80% disk reads. It also achieves up to 97% content deduplication in the host-cache.

2 Affinity-aware Modeling of CPU Usage for Virtualized Applications

An important consideration for migration and consolidation related decisions is the VM's resource requirement on the target host. VMs that mutually communicate are said to have *network affinity* and when VMs migrate, they may get colocated or dispersed w.r.t each other—resulting in different CPU overheads for network communication. Network traffic between colocated VMs is considered *intra-PM*, whereas between dispersed VMs is *inter-PM*. Given a communicating VM pair, migration of one may cause change of network traffic between them from *inter-PM* to *intra-PM*, or vice versa, depending on whether they get colocated or dispersed, respectively—this is referred to as the *mutable* nature of network affinity. In this work, we build *affinity-aware* models to predict expected CPU requirements upon colocation or dispersion of VMs. We make the following contributions,

- *Event profiling* of intra-PM and inter-PM network communication paths in Xen.
- *Benchmarking* of CPU usage of VMs (Xen and KVM) for various workloads in colocated and dispersed configurations.
- Development of *affinity-aware pair-wise* models to predict *total* as well as *differential* CPU usage, when a pair of VMs move between dispersed and colocated configurations.
- Apply the pair-wise models to *multi-VM scenarios* to predict CPU usage of a set of VMs.
- *Comprehensive evaluation* using synthetic workloads & benchmark applications, for pair-wise models as well as multi-VM scenarios.

2.1 Micro-benchmarking the effect of colocation on CPU Usage

It is discussed in [3] that colocated provisioning can result in changes in resource usage—however, empirical quantification is lacking. We address the following questions, (i) For communication between dispersed VMs, what is the change in CPU usage when VMs get colocated? (ii) For other network traffic and workloads, is colocated CPU usage a summation of dispersed usages?

Setup: We developed a multi-threaded load generation tool, `LoadGen`, that generates CPU-intensive, network-intensive, disk-intensive and mixed workloads. We performed benchmarking for both Xen and KVM. Fig. 1 shows the experimental setup that we used—two PMs host the VMs, and each PM is connected via a Layer-2 Switch to an NFS server which hosts the VMs' virtual disk images. The Layer-2 Switch and all network links of the machines operate at 100 Mbps. The Controller uses scripts to automate load generation and resource-usage measurements.

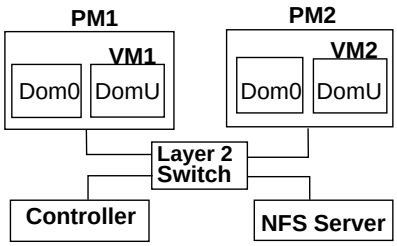


Fig. 1: Setup for benchmarking, profiling and model evaluation.

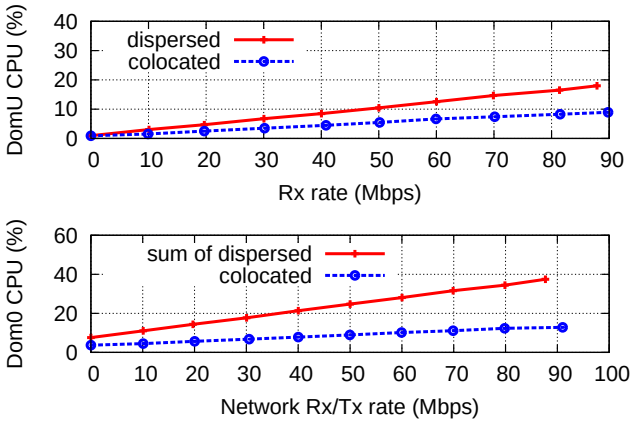


Fig. 2: CPU utilization due to *mutable* network traffic (in Xen setup).

Benchmarking results: Fig. 2 plots DomU and Dom0 CPU utilization for varying network usage between a pair of VMs (i.e., *mutable* traffic), in colocated and dispersed scenarios. At 90 Mbps, the colocated DomU utilization is 10%, which is almost half of the dispersed case. Further, at 40 Mbps and 90 Mbps, the difference in Dom0 CPU usage between dispersed and colocated scenarios is 14% and 25%, respectively.

From our benchmarking, we concluded that differences in CPU usage across colocated and dispersed placements occurs only for VMs that are mutually communicating, i.e., *mutable* traffic. For workloads like CPU/disk workloads and *immutable* traffic, different placements do not affect CPU usage. Further, resource utilization and Dom0/DomU CPU utilization are linearly related.

2.2 Linear regression modeling for CPU requirement estimation

Colocation and Dispersion models: We develop pair-wise CPU estimation models that can predict total CPU requirement in target scenario based on source scenario's resource usages. Specifically, using resource usage measurements of dispersed case, the "colocation" model predicts CPU utilization for colocated scenario, and based on resource usage in colocated case, the "dispersion" model predicts for dispersed scenario.

Our benchmarking revealed that only the mutable network usage causes changes in CPU requirement, upon a VM migration. Hence, we build models to use two approaches of prediction:- (i) Predict *total* CPU requirement based on multiple resource usage profiles—CPU, disk and network, (ii) Predict *differential* CPU requirement based only on mutable network traffic metrics—later, take summation of prediction with the source scenario's CPU usage to estimate the total CPU requirement. Next, we briefly explain both the approaches.

Approach 1: Prediction of total CPU requirement

Intuition: The total CPU requirement of a virtual machine accounts for all its activities, including usage of all other resources. Hence, this model has all resource metrics as its parameters: (i) 4 metrics for mutable and immutable transmit and receive network rates, (ii) 3 CPU metrics of `iowait`, `system` and `user` CPU, and (iii) 4 metrics for disk read/write rates.

Method: We use multi-linear regression modeling, wherein the relation between estimated CPU and resource parameters is captured, as shown in Eqn. (1).

$$\text{CPU}^i_{estimated} = C_0 + C_1 \times M^i_1 + C_2 \times M^i_2 + ... + M^i_m \tag{1}$$

where, i is an iterator over each sample point, m is the number of parameters, M^i_j is the value of parameter M_j collected in the sample point number i, and $\text{CPU}^i_{estimated}$ is the CPU usage either after dispersion or after colocation of VMs. To solve for the $m+1$ coefficients C_k, we use Robustbase package, part of the R statistical computing environment.

Approach 2: Prediction of differential CPU requirement

Intuition: The difference in CPU usage upon transition between colocated and dispersed placements of a VM pair, depends only on the mutable network affinity between them. Hence, we can predict only the difference, and sum it with the original scenario's CPU usage, to obtain the total CPU requirement. The model parameters are bit-rates (in Kbps) and packet-rates (in packets per second) for both transmission (Tx) and reception (Rx), thus four parameters in all.

Method: Formally, Eqn. (2) captures the general notion of change in CPU usage while transitioning between colocated and dispersed scenarios.

$$\Delta\text{CPU} = \text{CPU}^{scenario1} - \text{CPU}^{scenario2} \tag{2}$$

where, scenario$\{1|2\}$ = $\{colocated|dispersed\}$. Each DomU and Dom0 model represented as

$$\Delta\text{CPU}_{est} = C_0 + C_1\text{MutRx}_{Kbps} + C_2\text{MutRx}_{p/s} + C_3\text{MutTx}_{Kbps} + C_4\text{MutTx}_{p/s} \tag{3}$$

where, "MutRx" and "MutTx" stand for mutable network receive and transmit traffic, respectively. Once again, the Robustbase package is used to solve for the model coefficients.

2.3 Applying the pair-wise models to multi-VM scenarios

In an example multi-VM scenario as shown in Fig. 3, migration of $VM2$ can cause it to be dispersed from $VM1$ and become colocated with $VM3$—we refer to this as a "combined" transition. Here, CPU requirement estimation for $VM2$ and $PM2$ needs a multi-phase prediction methodology. We need to first apply the *dispersion model* to the measured metrics corresponding to network-affinity level between $VM2$ and $VM1$, to get an intermediate estimate, and then apply the *colocation model* to the intermediate estimate based upon network-affinity level between $VM2$ and $VM3$ to get the final estimate.

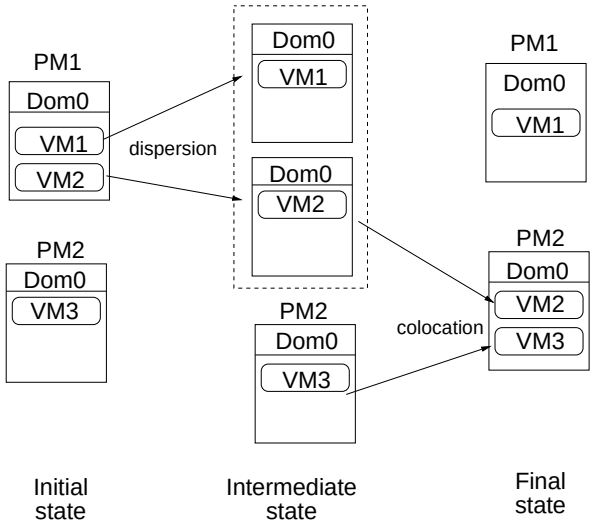


Fig. 3: Combined transition for $VM2$

Num of clients	Max n/w traffic (Mbps)	Max error(%CPU)		
		DomU	Dom0 colo	Dom0 disp
500	6.8	1.73	1.33	0.90
1000	13.4	1.50	0.98	0.82
1500	19.8	1.42	0.69	0.90
2000	26.4	0.90	1.08	1.21
2500	32.6	1.03	1.29	1.27
3000	39.4	1.36	1.50	1.54

Table 1: Model accuracy with varying load

2.4 Summary of Experimental evaluation

We generated CPU, disk, network and combinational workloads of randomly picked levels to form our testing dataset. Further, we used an application benchmark (RUBiS) to perform testing—we generated different levels of network traffic by using different number of clients in each workload. Finally, we conducted experiments with multi-VM scenarios wherein every VM migration results in a different placement configuration, and used the multi-phase prediction methodology to predict for every VM and PM. Here, we present a list of main results.

1. For both the approaches 1 and 2, we performed evaluation with synthetic datasets and observed that Approach 1 (prediction of total CPU) had higher error of 4-5% as compared to Approach 2 (prediction of differential CPU) error of 2%.
2. We conducted experiments with various number of clients in RUBiS [4] and found maximum error in each case to be within 2% absolute CPU (refer Table 1).
3. We created a 3-tier RUBiS setup and used the multi-phased prediction methodology to predict the CPU usage of VMs and PMs across multiple transitions, and found maximum estimation error within 2% absolute CPU.

2.5 Conclusions

In this work, we performed network benchmarking, to quantify the effects of network affinity on CPU usage when VMs are colocated versus dispersed. Next, we developed VM *pair-wise* models that can estimate "colocated" CPU usage, on being input their individual dispersed-case resource usages, and to estimate "dispersed" CPU usage based on colocated-case resource usages, using two approaches. Experiments revealed that Approach 2 provides better estimates with maximum error less than 2%. Finally, we demonstrated the application of *pair-wise* models to predict for multi-VM scenarios, with high accuracy.

3 DRIVE: Using Implicit Caching Hints to Achieve Disk I/O Reduction in Virtualized Environments

Content similarity within a single VM image is called intra-VM similarity and across multiple VMs is referred to as inter-VM similarity. A study of content similarity amongst 525 VM images from a production environment [5] reported 30% blocks repeating at least twice and 12% blocks repeating 5 times. A study in [6] reported that up to 90% of total similarity observed among VM disk content is intra-VM, rather than inter-VM. The work in [7] studied the degree of content similarity in application workloads (web, mail and file system) hosted in VMs, and reported that there are multiple blocks having identical content within each application workload.

A VM's virtual disk is present on the host's storage, which may be a local disk or a network-attached remote disk. Disk access times are in the order of a few milliseconds [8, 9], whereas cache access timings are orders of magnitudes lower [10, 11]. Hence, improving caching effectiveness can help improve storage access performance and overall application performance. However, due to content similarity across blocks, host cache effectiveness is limited by two factors: (i) duplicate I/O problem—multiple blocks containing same content being fetched from disk, and (ii) duplicate content problem—multiple blocks in cache have same content. Both problems can be addressed by maintaining content similarity metadata by actively tracking insertion and eviction of blocks in page cache, however this would require invasive changes in the host kernel. We develop a read I/O redirection technique positioned within the VM's virtual block device, such that both problems are addressed. If a block of content is present in the host's page-cache, it need not get fetched from disk. This, in turn, ensures that multiple blocks with same content are (almost) never stored into the host's page-cache, i.e., a content-deduplicated page-cache is achieved. Thus, we reduce disk I/O without actively tracking page-cache state, as well as without maintaining any explicit content cache. The **contributions** of this work are,

1. Simulation-based analysis of existing system [7] to show that it is inefficient.
2. Design and implementation of the DRIVE system that tackles both the duplicate content and duplicate I/O problems,
3. Trace-based evaluation of DRIVE with prototype implementation in a custom simulator.

3.1 Analysis of existing I/O deduplication technique: IODEDUP

Harnessing content similarity to avoid duplicate disk I/O requests is called I/O deduplication. The work in [7] builds metadata regarding block content similarity, and maintains a content-cache—retains a single copy of each content. Fig. 4 shows I/O Deduplication (henceforth IODEDUP) system architecture. When read requests encounter a block-cache miss, they are inter-

Trace Name	Num of read requests	Num of write requests	Max share factor	Max occur factor	%blks having 2 copies	%content occuring twice	Write intensivity factor
homes	4,052,176	17,110,222	5904	5905	5	10	4.2
mail	2,375,409	18,007,471	57015	57015	5	6	7.6
webvm	3,116,456	11,177,702	124	125	35	45	3.6

Table 2: Summary statistics of traces used for evaluation

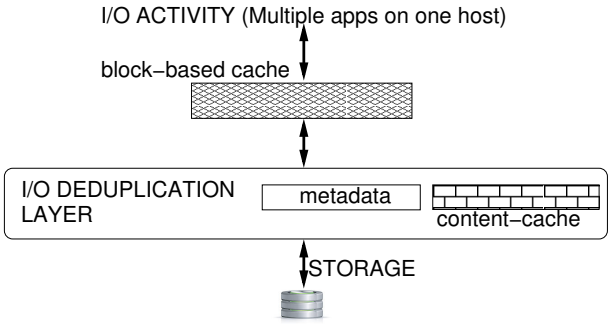


Fig. 4: System Architecture of IODEDUP

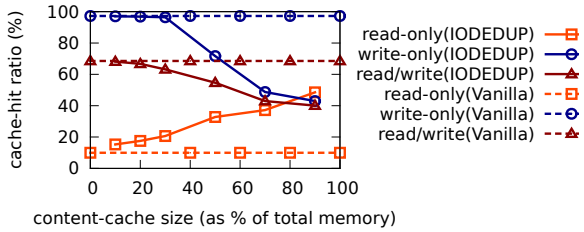


Fig. 5: Hit ratios in IODEDUP upon *webvm*; total cache size is 512 MB.

cepted by IODEDUP system and serviced from content-cache, if possible—thus avoiding duplicate content fetches from disk. However, since only a fraction of total space can be reserved as content-cache, the duplicate content problem persists. Secondly, since the content-cache partakes from total memory space available, optimally sizing the content-cache is essential because content-cache benefits may be application or request-mix specific.

Simulation setup: We built a custom simulator, `SimReplay`, with prototype implementations of Vanilla and IODEDUP systems. In Vanilla system, a read request's block number is first looked up into block-cache. If not found in cache, the block is fetched from disk. In IODEDUP system, the disk fetch path due to block-cache miss is intercepted and metadata is used to lookup the block in content-cache. A content-cache hit averts the disk fetch, however, a miss necessitates disk fetch and metadata update. For exploration of IODEDUP system's effectiveness, we chose six content-cache size settings as 10%, 20%, 30%, 50%, 70% and 90% of the total memory size, and remaining as the block-cache size.

Traces for simulation: We borrow traces provided via [7] which are over 21 days from three production systems: (i) webvm–from virtualized web-servers hosting webmail proxy and course management system, (ii) mail–email server traces, and (iii) homes–file server traces. Table 2 presents a summary of the traces.

Analysis results: Fig. 5 shows cache-hit ratios for webvm trace for read-only, write-only and read-write replays. IODEDUP has different optimal content-cache sizes for different cases—90% for read-only, 10% for write-only, and 10% for read/write. Further, for read-write trace, performance of IODEDUP worsens compared to Vanilla, as content-cache size increases. Hence,

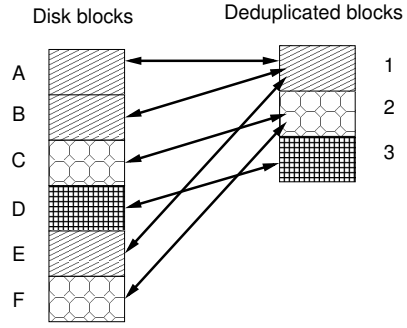


Fig. 6: Metadata store semantics: *each block points to a unique deduplicated block, and each deduplicated block reverse maps to multiple actual blocks*

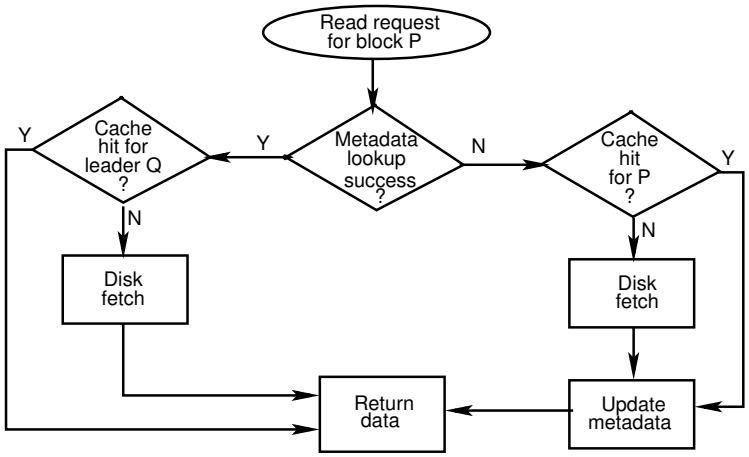


Fig. 7: Flow path(s) for read in DRIVE: *P is requested block, Q is corresponding "leader" block.*

we consider 10% as optimal content-cache size for further evaluation.

3.2 DRIVE system requirements and design

We build an I/O redirection system positioned above the block-cache which uses implicit caching hints to perform the redirection. The **system requirements** of an efficient I/O reduction system:

1. Fingerprinting mechanism to identify content similarity across different blocks.
2. Maintaining implicit caching hints within metadata to aid future I/O redirection.
3. Interception of block read request path for metadata lookup and I/O redirection, if present.
4. Interception of block read return path for metdata update, if not previously present.
5. Interception of block write request path for metadata invalidation.

System Design. The DRIVE system has three main components: (i) *Metadata maintenance*, (ii) *Maintaining implicit hints regarding host-cache state*, (iii) *Hint-based read I/O redirection*.

(i) Metadata maintenance: For read requests, we compare content fingerprints to determine whether it is "new" or "duplicate", and update metadata accordingly. For write requests, we invalidate existing mappings for those blocks, so that stale metadata is not used for the next read request redirection. The semantics of metadata store is as illustrated in Fig. 6.

(ii) Maintaining hints regarding host-cache state: To avoid invasive tracking of host-cache (i.e., by trapping and intercepting each insertion and eviction within cache), we use *implicit hints* for redirection instead. When a read request is serviced, the corresponding physical block is in host-cache, and if found to be identical to any previously requested block, we record the current block ID as "leader", for that content. The recording as leader is done to aid redirection of future I/O requests that request the same content, and is a hint regarding the state of host-cache.

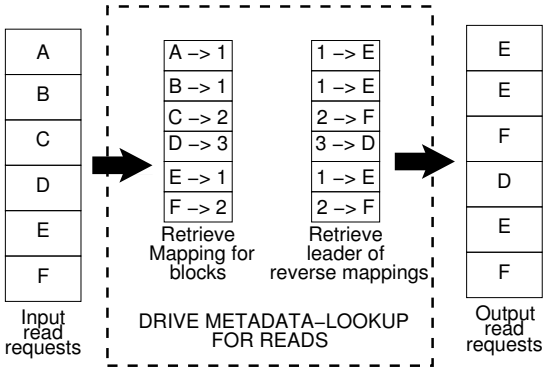


Fig. 8: Read request redirection in DRIVE

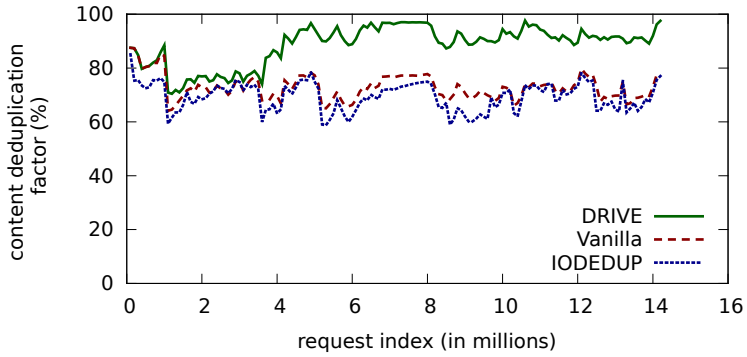


Fig. 9: Host cache effectiveness upon *webvm* trace.

(iii) Read I/O redirection using implicit hints: Before a read request hits the host-cache, the DRIVE system intercepts it and looks up metadata to check for a recorded leader. If present, original block ID in request is replaced with the leader block ID. Fig. 7 depicts the multiple flows in the read path, wherein an incoming request for block P is looked up in metadata store and if successful, redirected to block Q.

Overall operation: Fig. 8 illustrates via an example. Suppose blocks A, B and E have identical content (mapped to deduplicated ID 1) and E is appointed their leader since it was fetched the latest. Whenever next, read requests for A, B or E are received, request is redirected to read block E. Note that, blocks A, B and C will be in cache the first time they are fetched but once they get evicted, they will never enter cache again, unless writes are done to them. This implies that the cache is operated in an almost fully content-deduplicated fashion, hence improving its effectiveness in reducing the number of disk reads.

3.3 Summary of experimental evaluation

For evaluation, we extend our simulator, and replay same traces as before. For every trace replay, measures like cache hits, cache misses, and disk fetches are recorded. We performed evaluation for individual (single-VM) workloads, as well as simultaneous (multi-VM) workloads. Here, we present a summary list of main results.

1. Our evaluation shows that with webvm workload, DRIVE has higher cache-hit ratios than both Vanilla and IODEDUP systems, 10% and 20% better, respectively. Moreover, DRIVE has higher number of "disk reads reduced", with nearly 85% improvement over Vanilla, and a 2.8× improvement over IODEDUP.

2. A classification of read request responses into compulsory misses, capacity misses and cache hits, for each trace, showed that *webvm* trace contained a lot of capacity misses in the Vanilla system, and DRIVE succeeds in reducing them to 5% of the Vanilla system.

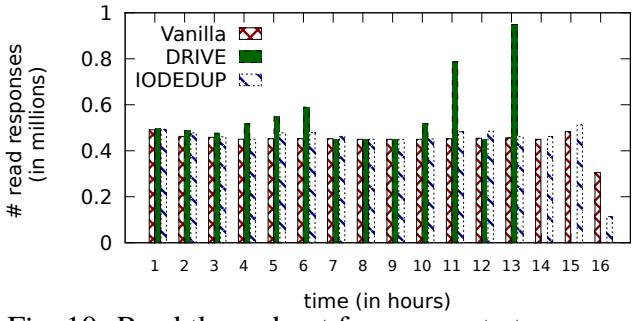


Fig. 10: Read throughput for aggregate trace

Scheme	Hit rate(%)	Reads saved(%)
Vanilla	61.2	1.6
DRIVE	67.6	18.5
IODEDUP	62.4	4.3

Table 3: Aggregate replay performance

3. We define content deduplication factor as the ratio of number of unique content blocks to total number of blocks in cache. Thus, higher the content deduplication factor, better is cache efficiency. Fig. 9 shows the variation in content deduplication factor, as each request of the webvm trace is replayed. We see that DRIVE system achieves significantly higher content deduplication than both Vanilla and IODEDUP—a content deduplication factor of up to 97%.

4. To perform evaluation of multi-VM workloads executed on a virtualized host, we aggregated the *homes* and *webvm* traces in timestamp order, and replayed it. The metrics tabulated in Table 3 show that although the cache-hit ratios are comparable in all three schemes, DRIVE system averts 18% disk reads as compared to 1.6% of Vanilla and 4.3% of IODEDUP. Fig. 10 plots the read response throughput per hour, assuming continuous replay of requests. We can see that DRIVE produces more number of responses per hour in the 4th, 5th, 6th hours and so on, hence resulting in earlier completion than Vanilla or IODEDUP.

3.4 Conclusions

In this section, we considered the problem of improving disk access performance by improving host cache efficiency. Typically, caches are referenced by block number, and can not recognize con- tent similarity across multiple blocks. Elimination of duplicate read I/O requests is referred to as I/O Deduplication. Existing work (IODEDUP) uses a split-cache approach [7], with a part of block-cache reserved as a content-cache. We demonstrated that a split-cache approach is inefficient, and presented the DRIVE system which performs I/O redirection to implicitly manipulate the whole underlying cache as a content-deduplicated cache. Only the VM's own disk access history is introspected to obtain implicit hints regarding host cache state, to be used for read I/O redirection.

We performed comparative trace-based evaluation in a custom simulator. The evaluations showed that, the DRIVE system achieves a high content deduplication factor of up to 97%. This is the key reason for better performance, with up to 20% higher cache-hit ratios, and up to 85% higher number of disk reads reduced than the Vanilla system.

4 Conclusions

In any virtualization environment, there are two types of resources involved:- (i) resources allocated to virtual machines, and (ii) resources available in host to enable and support the virtualization. In our thesis, we address two problems related to improving the resource usage efficiency of resources in virtualized environments.

To address the problem of server consolidation or migration to a target host, the foremost requirement is to estimate the VM's expected resource requirements on the target host. This can enable to decide whether such a migration is advisable. In the first problem, we address the estimation of CPU resource requirements of mutually communicating VMs as they transition between colocated and dispersed placements. In Xen virtualization environment, since the privileged domain Dom0 needs CPU scheduling to perform IO operations (network operations and disk operations for NFS-mounted VM images) on behalf of the VMs, we are also interested in predicting the corresponding Dom0 CPU resource requirement. Though we demonstrate our work with Xen, our approach is generic and is applicable to other virtualization solutions too.

Two VMs are said to have *network-affinity* if they communicate with each other, and we show that colocation of communicating VMs on a single PM helps reduce the total CPU usage. With detailed measurements, we justified the need for affinity-aware placement and presented our approach to estimate CPU requirement for colocated and dispersed VMs. We built models to predict the absolute usage as well as the difference in usage, and observed that the prediction of difference is more accurate, achieving **maximum error within 2% absolute CPU usage**. Further, we applied the pair-wise prediction models using a multi-phase prediction methodology to predict for multi-hop VM transition scenarios as well.

Second, we consider the problem of improving virtual disk access performance by improving host cache efficiency. Typically, caches are referenced by block number, and can not recognize content similarity across multiple blocks. Hence, the system ends up storing multiple copies of same content into cache. Elimination of duplicate read I/O requests is referred to as I/O Deduplication. Existing work uses a split-cache approach, with a part of the block-based cache reserved as a content-based cache. We demonstrated that a split-cache approach is inefficient, and presented the DRIVE system which performs I/O redirection to *implicitly* manipulate the whole underlying cache as a content-deduplicated cache. Only the VM's own disk access history is introspected to obtain implicit hints regarding host cache state, to be used for read I/O redirection.

We performed comparative trace-based evaluation in a custom simulator. The evaluations showed that, the **DRIVE system achieves a high content deduplication factor of up to 97%**. This is the key reason for better performance, with up to 20% higher cache-hit ratios, and up to 80% higher number of disk reads reduced than the Vanilla system.

References

[1] The AWS Community. Amazon Elastic Compute Cloud (Amazon EC2). Website. http://aws.amazon.com/ec2/. Accessed Apr 24, 2015.

[2] Ubuntu Community. Bootstack—Your Managed Cloud. Website. http://www.ubuntu.com/cloud/managed-cloud. Accessed May 11, 2015.

[3] Jason Sonnek and Abhishek Chandra. Virtual Putty: Reshaping the Physical Footprint of Virtual Machines. In *Proceedings of the Workshop on Hot Topics in Cloud Computing*, June 2009.

[4] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and Implementation of Dynamic Web Site Benchmarks. Proceedings of the 5th Workshop on Workload Characterization, 2001.

[5] K. R. Jayaram, Chunyi Peng, Zhe Zhang, Minkyong Kim, Han Chen, and Hui Lei. An Empirical Analysis of Similarity in Virtual Machine Images. In *Proceedings of the Middleware Industry Track Workshop*, pages 6:1–6:6, 2011.

[6] Sean Barker, Timothy Wood, Prashant Shenoy, and Ramesh Sitaraman. An Empirical Study of Memory Sharing in Virtual Machines. In *Proceedings of the USENIX Conference on Annual Technical Conference (ATC)*, pages 25–25, 2012.

[7] Ricardo Koller and Raju Rangaswami. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 211–224, 2010.

[8] Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. In *Proceedings of the 7th International Conference on World Wide Web (WWW)*, pages 107–117, 1998.

[9] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, pages 18:1–18:14, 2008.

[10] Robert Love. Linux Kernel Development Second Edition. http://www.makelinux.net/books/lkd2/ch15. Accessed Jan 5, 2015.

[11] Grzegorz Miłós, Derek G. Murray, Steven Hand, and Michael A. Fetterman. Satori: Enlightened Page Sharing. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, 2009.

Publications List

Conference Proceedings

1. *DRIVE: Using Implicit Caching Hints to achieve Disk I/O Reduction in Virtualized Environments*. Proceedings of the 21st International Conference on High Performance Computing (HiPC), 2014. Sujesha Sudevalayam, Purushottam Kulkarni, Rahul Balani and Akshat Verma.

2. *Affinity-aware Modeling of CPU Usage for Provisioning Virtualized Applications*. Proceedings of the 4th International Conference on Cloud Computing (CLOUD), 2011. Sujesha Sudevalayam and Purushottam Kulkarni.

Journal Publications

1. *Affinity-aware Modeling of CPU Usage with Communicating Virtual Machines*. Journal of Systems and Software (JSS), 2013. Sujesha Sudevalayam, Purushottam Kulkarni.

2. *Energy Harvesting Sensor Nodes: Survey & Implications*. IEEE Communications Surveys and Tutorials 2011. Sujesha Sudevalayam and Purushottam Kulkarni.

Technical Reports

1. *CONFIDE: Content-based Fixed-sized I/O Deduplication*. Technical Report, IIT Bombay. Sujesha Sudevalayam, Purushottam Kulkarni, Rahul Balani and Akshat Verma. IITB/CSE/2014/April/60, TR-CSE-2014-60.

2. *Affinity-aware Modeling of CPU Usage for Provisioning Virtualized Applications*. Technical Report, IIT Bombay. Sujesha Sudevalayam and Purushottam Kulkarni. IITB/CSE/2011/February/34, TR-CSE-2011-34.

3. *Colocation-aware Modeling of CPU Usage for P2V Transitioning Applications*. Technical Report, IIT Bombay. Sujesha Sudevalayam. IITB/CSE/2014/March/59, TR-CSE-2014-59.

4. *Balancing Response Time and CPU allocation in Virtualized Data Centers using Optimal Controllers*. Technical Report, IIT Bombay. Varsha Apte, Purushottam Kulkarni, Sujesha Sudevalayam and Piyush Masrani. IITB/CSE/2010/April/29, TR-CSE-2010-29.

5. *Energy Harvesting Sensor Nodes: Survey and Implications*. Technical Report, IIT Bombay. Sujesha Sudevalayam and Purushottam Kulkarni. IITB/CSE/2008/December/19, TR-CSE-2008-19.