Resource Provisioning and Usage Optimization in Virtualized Environments

Sujesha Sudevalayam

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
{sujesha}@cse.iitb.ac.in

January 13th, 2015
Pre-synopsis Presentation

Pay-Per-Use Service Model

Electricity Grid

Public Transport

- Software as a Service
- Platform as a Service
- Infrastructure as a Service

Virtualization ← Enabling technology

Pay-Per-Use Service Model

Electricity Grid

Public Transport

- Software as a Service
- Platform as a Service
- Infrastructure as a Service

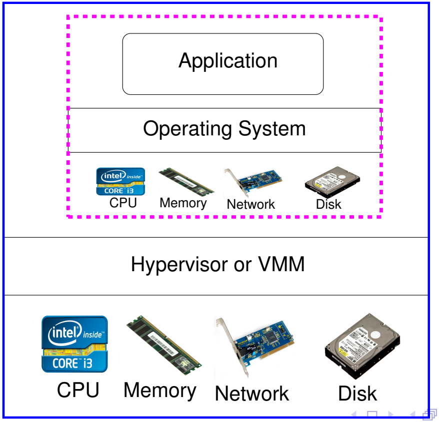
Enabling technology

Virtualization

Thesis Scope

Two types of resources in virtualized environment

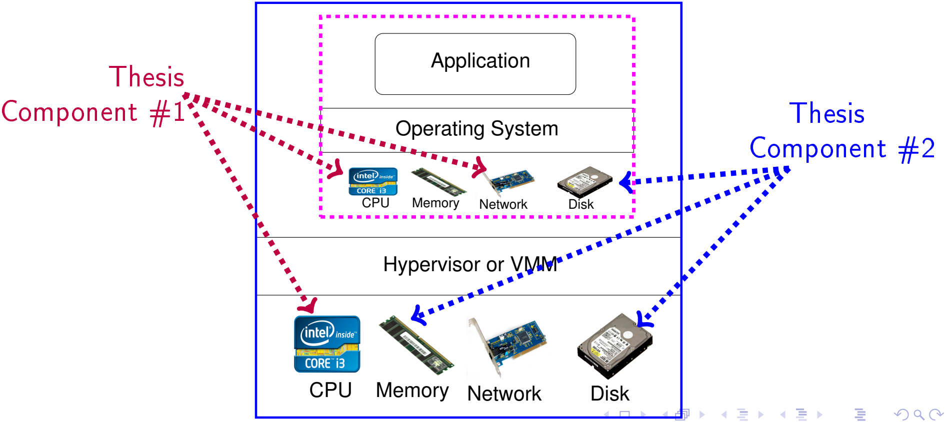
1. Resources allocated to virtual machines
2. Resources available to host for virtualization operation & overheads



Thesis Scope

Two types of resources in virtualized environment

1. Resources allocated to virtual machines
2. Resources available to host for virtualization operation & overheads



Top 3 Contributions

Contribution 1: Network-affinity aware CPU Usage Estimation

- *Initial Attempt:* Linear model to predict “total” CPU requirement
- *Challenge:* Maximum error 5-6% absolute CPU
- *How Overcame:* Predict “differential” CPU usage—Max error 1-2%

Contribution 2: VM Disk I/O Reduction by Host-cache Manipulation

- *Initial Attempt:* Performing variable-sized deduplication
- *Challenge:* Real-world trace available only for fixed-size, not variable
- *How Overcame:* Show value of caching hints in fixed-size dedup

Contribution 3: I/O trace characterization for deduplication

- *Initial Attempt:* I/O tracing toolkit but no production tracing
- *Challenge:* Need real workloads/realistic traces for characterization
- *How Overcame:* Extensive dataset survey to make the *case* that need to generate realistic I/O traces with content representation

Content Outline—Part I

Affinity-aware CPU usage estimation in *migratory* VM scenarios

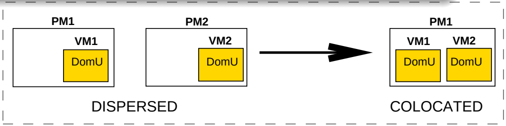
1. Profiling study of Xen network virtualization to show the different flow paths for *intra-PM* and *inter-PM* network traffic
2. Benchmarking of CPU usage for various workloads in colocated and dispersed scenarios (demonstrated to be linear)
3. Pair-wise linear regression model to predict total CPU when network traffic changes nature between intra-PM and inter-PM
4. Pair-wise linear regression model to predict differential CPU usage
5. Application of pair-wise models to predict for multi-VM scenarios

Tools and deliverables

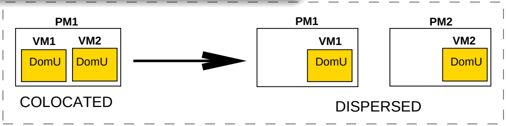
- `WLoadGen`: A load generator for CPU, disk & network loads

Migration-Enabled Resource/Performance Management

Consolidate/colocate VMs for Resource Efficiency

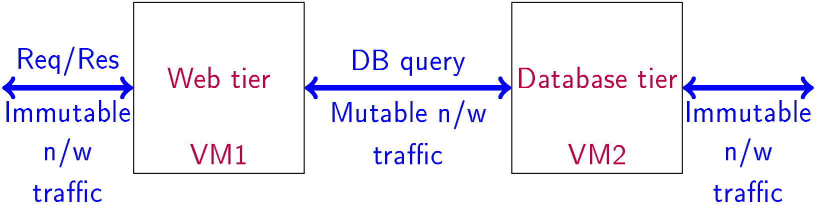


De-consolidate/disperse VMs for QoS



- Both colocation and dispersion need resource usage estimation
- Incorrect estimation is sub-optimal
  - Under-estimation => degraded performance
  - Over-estimation => wasted resources

Mutable and Immutable Network traffic for *Migratory* VMs



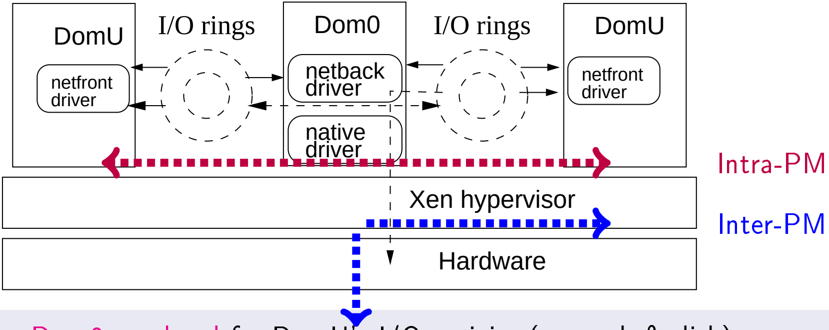
Mutable n/w traffic

Network traffic between VMs whose relative placement may *change between colocated and dispersed*, due to server consolidation strategies

Our hypothesis

Mutable network traffic has different CPU overheads in colocated and dispersed scenarios => ignoring affinity effects could result in incorrect CPU usage estimation

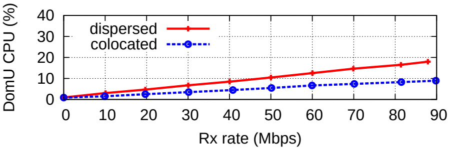
Communicating VMs (Xen-view)



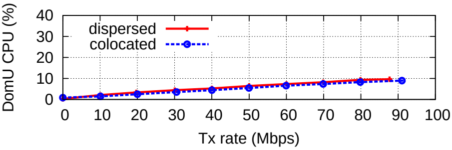
- Dom0 overhead for DomU's I/O activity (network & disk)
- Intra-PM network traffic
  - Dom0 does not use native I/O drivers
  - Shared memory based copying of packets
- Less CPU overhead for *intra-PM* traffic compared to *inter-PM*
- Needs to be accounted for during VM migration

Benchmarking: Effect of colocation on CPU usage for *Mutable* N/w traffic

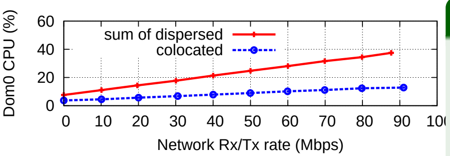
Benchmarking setup: 2 VMs on 2 PMs—dispersed and colocated scenarios
Network load: Transmitted (Tx) by one VM and Received (Rx) by other



(a) Receiving DomU CPU util



(b) Transmitting DomU CPU util



(c) Dom0 CPU util for Rx/Tx

Observations

- **DomU**: Rx increase from 20-90 Mbps =>decrease of 2-8% CPU util
- **Dom0**: Increase from 20 to 90 Mbps =>decrease from 9-25% CPU util

Benchmarking: Effect of colocation on CPU usage for Immutable n/w traffic, CPU and disk loads

Benchmarking setup: 4 VMs on 4 PMs—dispersed and colocated scenarios

Table: Percentage CPU usage for Immutable Rx

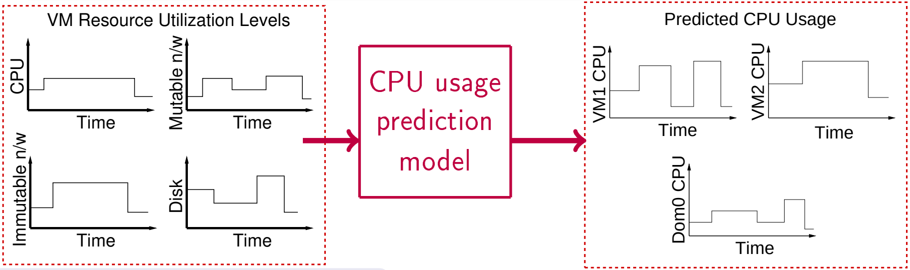
Immutable Rx (Mbps)	% CPU utilization	
	Dispersed case $VM_1, VM_2, \sum Dom0_i$	Colocated case $VM_1, VM_2, Dom0$
<20, 50>	4, 7, 18	4, 7, 14
<40, 10>	6, 2, 15	6, 2, 11
<60, 10>	8, 2, 18	8, 2, 14

Observations

1. No change in DomU CPU usage between colocated and dispersed
2. Dom0 CPU usage change of 4% for extra Dom0 instance (constant)
3. Similar observations for other workloads—CPU and disk read/write

Problem: Affinity-aware Resource Requirement Estimation

Given a pair of VMs and their resource utilization levels, predict the CPU resource requirement of DomU & Dom0, when VM placement scenario changes between dispersed and colocated.



For VM1 and VM2 (dispersed)

(colocated)

Core Idea

Since correlation of CPU usage with all other resources usage is linear, build linear prediction models

Linear Regression Modeling for CPU Estimation

Parameters in the models

- **CPU** metrics: user, system, iowait
- **Disk** metrics: read blocks/second, write blocks/second
- **Mutable and immutable** network metrics: Rx and Tx Kbps

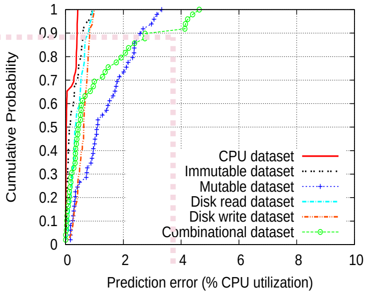
DomU Models

$$CPU_{colocated} = f(CPU, Disk, Mutable, Immutable)_{dispersed}$$
$$CPU_{dispersed} = f(CPU, Disk, Mutable, Immutable)_{colocated}$$

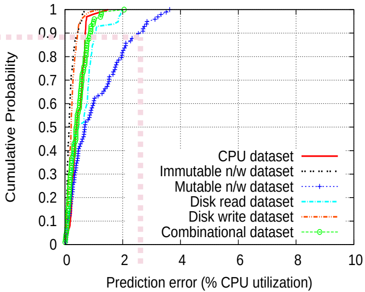
Dom0 Models

$$CPU_{colo} = f(CPU_1, Disk_1, Mutable_1, Immutable_1, CPU_2, Disk_2, Mutable_2, Immutable_2)_{disp}$$
$$CPU_{disp} = f(CPU_1, Disk_1, Mutable_1, Immutable_1)_{col}$$

Prediction for Synthetic workloads - Xen Dom0 model



(a) Dispersed to colocated

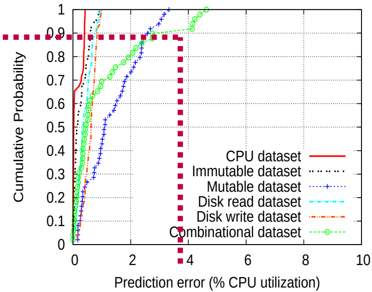


(b) Colocated to dispersed

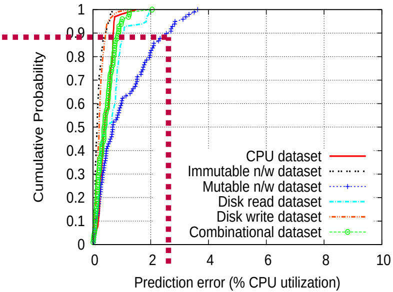
Observations

90^{th} percentile prediction error within 3% absolute CPU utilization, and maximum error 5-6% absolute CPU (Similarly for RUBiS workload as well)

Prediction for Synthetic workloads - Xen Dom0 model



(a) Dispersed to colocated



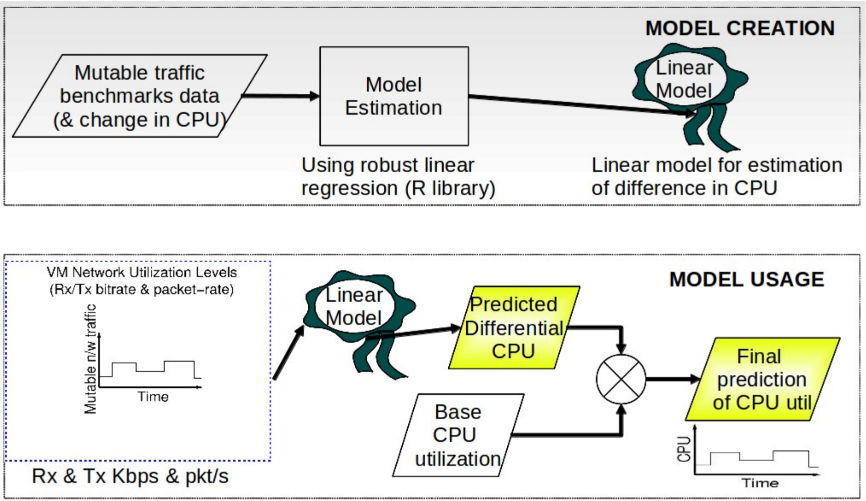
(b) Colocated to dispersed

Observations

90^{th} percentile prediction error within 3% absolute CPU utilization, and maximum error 5-6% absolute CPU (Similarly for RUBiS workload as well)

Building an Enhanced Prediction Model

Because “differential” CPU usage is only due to mutable n/w traffic



Evaluation of Differential CPU Prediction Models

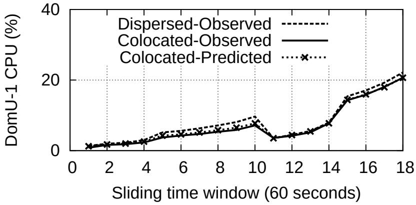


Figure: Colocated DomU-1 (Synthetic)

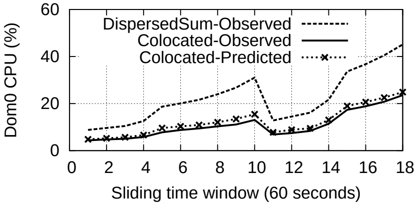


Figure: Colocated Dom0 (Synthetic)

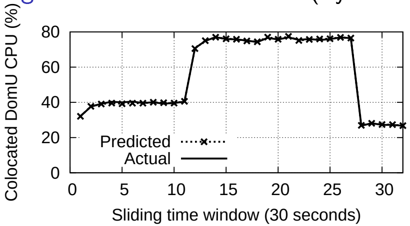


Figure: Colocated DomU-1 (RUBiS)

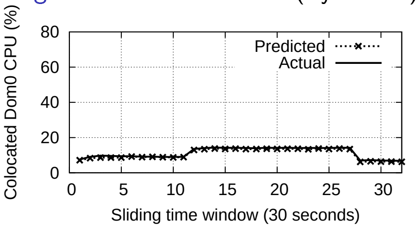


Figure: Colocated Dom0 (RUBiS)

Result

Maximum prediction error between 1-2% absolute CPU utilization.

Applying Pair-wise Models to Multi-VM Scenarios

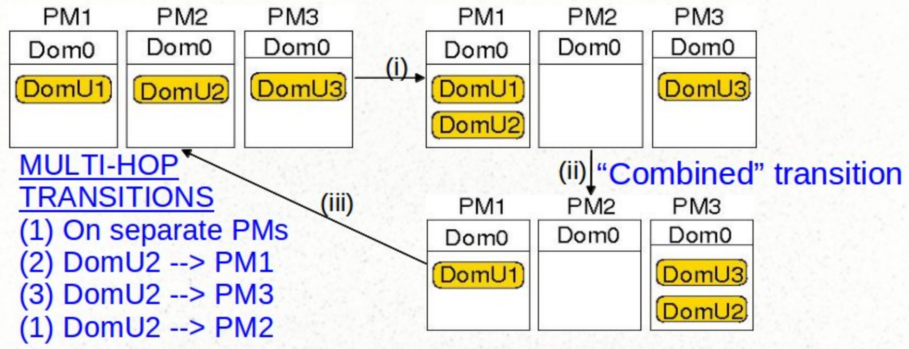


Table: Maximum error in Dom0 CPU utilization prediction

Transition	Max error (% absolute CPU)		
	Dom0-PM1	Dom0-PM2	Dom0-PM3
Transition (i)	0.75	-	-
Transition (ii)	1.99	-	0.85
Transition (iii)	-	0.51	0.43

Summary of Part I

- **Colocation of mutually-communicating VMs impacts their CPU requirement**
  - DomU: For Rx, increase from 20 to 90 Mbps => decrease from 2% to 8% CPU requirement
  - Dom0: Increase from 20 to 90 Mbps => decrease from 9% to 25% CPU requirement
- **Simple linear model shown to predict “differential” CPU requirement from mutable n/w traffic profiles**
  - Synthetic workloads: Max error within 1.5% absolute CPU utilization for both DomU and Dom0 models
  - RUBiS benchmark application: Max error within 1.5% for Web and DB tiers, and Dom0
  - Multi-VM scenario: Max error within 2% for all transitions

Content Outline—Part II

Part II. Host cache usage optimization for virtualized services

1. Analysis of existing work (IODEDUP) to show inconsistent performance
2. Redirection of I/O requests from within the virtual machines
3. To implicitly manipulate host cache in content-deduplicated fashion
4. Using implicit caching hints
5. Evaluation using public dataset available online
6. Case for generation of realistic I/O deduplication benchmarks

Tools and Deliverables

1. `SimReplay`: A simulator for analyzing host cache effectiveness
2. `preadwritedump`: A kernel module for I/O request tracing

Effect of Data Similarity on Host-cache Effectiveness

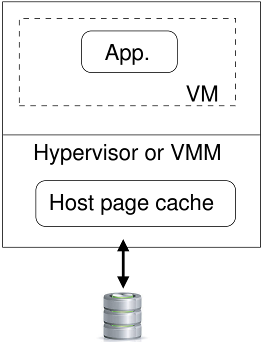


Figure: Typical virtualized system

Two optimization avenues

1. Duplicate I/O
2. Duplicate content in cache

Two orthogonal solutions

1. I/O deduplication (IODEDUP[1]) : but causes cache inclusiveness problem
2. Memory deduplication (Satori[2]) : dedupes after data is fetched

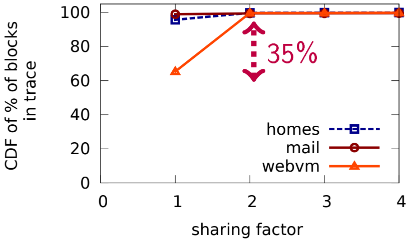
Sources of data similarity

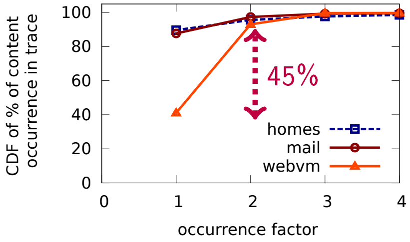
Similar operating systems, libraries, binaries, file copies, etc.

Aim of this work

Improve host-cache effectiveness *using* I/O deduplication techniques, i.e., *achieve both* in one stroke.

Traces[1] used for evaluation: Similarity study





Observations

- *homes* & *mail* traces have 95% blocks with sharing factor 1, whereas *webvm* trace has 35% blocks with sharing factor 2
- In *webvm* trace, 45% content occur twice, compared to 6-10% in *homes* and *mail* traces

Conclusions

webvm trace is likely to benefit the most from I/O deduplication

[1] Workload traces borrowed from the IODEDUP paper [1]. Traces available online at [3] and SNIA

Existing[2] I/O deduplication technique: IODEDUP [3]

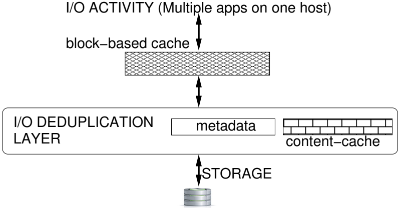


Figure: System Architecture of IODEDUP

Functioning

- Creates and maintains content-based cache
- Intercepts read requests & services without accessing disk if possible

[2] Other related work for I/O deduplication & reduction discussed in report.

[3] *I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance*

Existing[2] I/O deduplication technique: IODEDUP [3]

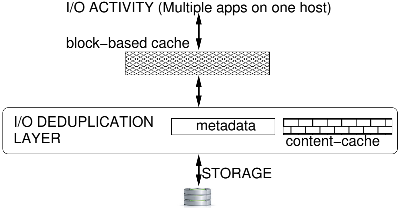


Figure: System Architecture of IODEDUP

Drawbacks

- Content-cache *sizing* needs exploration
- Block-cache still faces *duplicate content* problem

Functioning

- Creates and maintains content-based cache
- Intercepts read requests & services without accessing disk if possible

[2] Other related work for I/O deduplication & reduction discussed in report.

[3] *I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance*

Existing[2] I/O deduplication technique: IODEDUP [3]

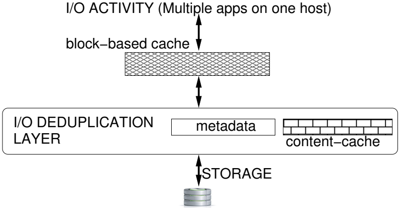


Figure: System Architecture of IODEDUP

Drawbacks

- Content-cache *sizing* needs exploration
- Block-cache still faces *duplicate content* problem

Functioning

- Creates and maintains content-based cache
- Intercepts read requests & services without accessing disk if possible

Our contribution

- Perform *study of cache effectiveness* for IODEDUP system, using a custom simulator

[2] Other related work for I/O deduplication & reduction discussed in report.

[3] *I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance*

Study of cache effectiveness for IODEDUP

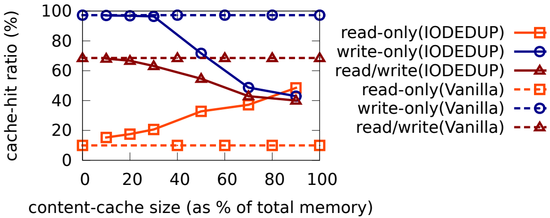


Figure: Cache-hit ratios for IODEDUP for *webvm* trace. Total cache 512 MB

Observations

1. Read-only trace has lowest performance at content-cache of 10% & highest at 90%
2. Write-only performance varies reverse, i.e., highest at 10% and lowest at 90%
3. At content-cache setting of 90%, read-only performance is 4× Vanilla, but read/write performance 42% worse than Vanilla.

Conclusion: Inconsistence in achievable cache effectiveness

Fundamental issues preventing efficient I/O reduction

Issues

1. In IODEDUP system [1] has cache inclusiveness problem
2. Memory deduplication [2] works after data is already fetched from disk

Obvious solution

- Operate host cache in fully-deduplicated fashion, such that only data not present in cache will be fetched from disk

Challenges in implementing obvious solution

1. Requires change to cache data structures and/or implementation to enable storing of content-based metadata
2. Requires metadata updates for every cache insertion
3. Requires invasive monitoring and metadata updates for every eviction from cache

DRIVE: Using implicit caching hints to achieve disk I/O reduction in virtualized environments

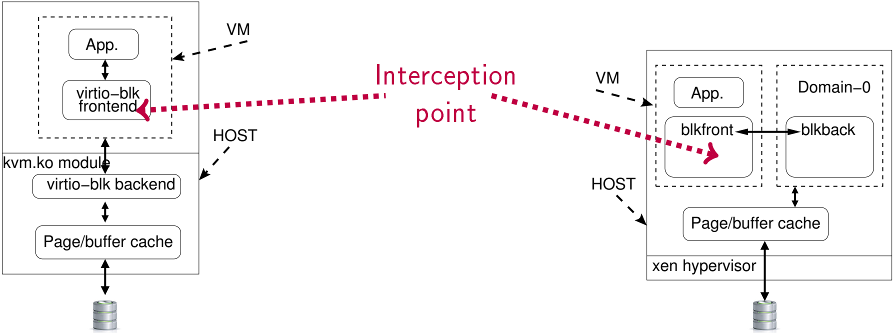
Our Approach

- Augment the virtual disk driver to **use implicit caching hints** to achieve an approximately fully-deduplicated host cache

System Requirements for DRIVE

1. Intercept block read *request* path for metadata lookup and I/O redirection, if present
2. Intercept block read *return* path for metadata update, if not previously present
3. Intercept block write *request* path for metadata invalidation
4. Maintain implicit caching hints within metadata to aid efficient I/O redirection.

Block request interception-point for DRIVE



(a) KVM split-driver architecture

(b) Xen split-driver architecture

Interception within VM's front-end driver

- De-coupling of the front-end and back-end drivers enables simple I/O redirection
- Results in implicit manipulation of host-cache as a content-deduplicated cache
- Exploits individual workload's content self-similarity, useful irrespective of co-hosted VMs
- Implementation within generic `virtio` drivers obviates dependence on VMM & guest OS

DRIVE metadata store: semantics and usage

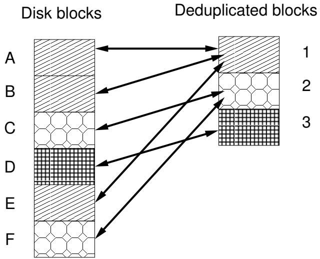


Figure: Semantics of metadata store.

DRIVE metadata store: semantics and usage

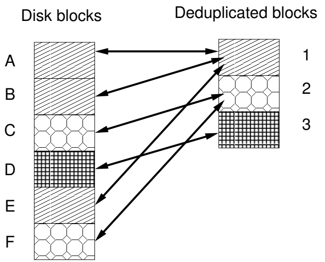


Figure: Semantics of metadata store.

Obtaining and using hints for I/O redirection

1. When a block is fetched, it is "known" to be cached
2. Above is noted in metadata, marked as *leader*
3. For next redirection, *leader* is used

DRIVE metadata store: semantics and usage

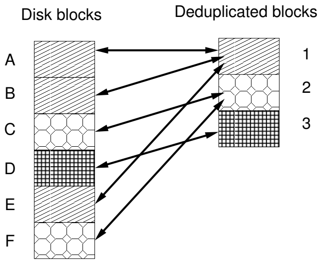


Figure: Semantics of metadata store.

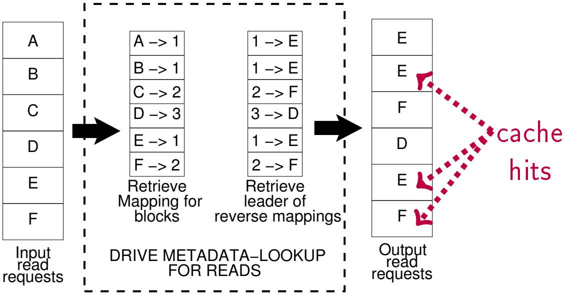


Figure: Example of read request redirection in DRIVE

Obtaining and using hints for I/O redirection

1. When a block is fetched, it is "known" to be cached
2. Above is noted in metadata, marked as *leader*
3. For next redirection, *leader* is used

Evaluating host-cache effectiveness in DRIVE system

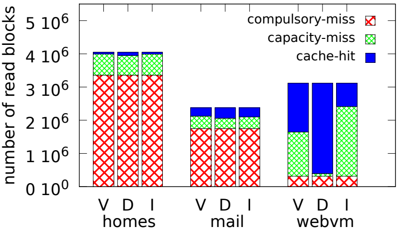


Figure: Classification of read responses

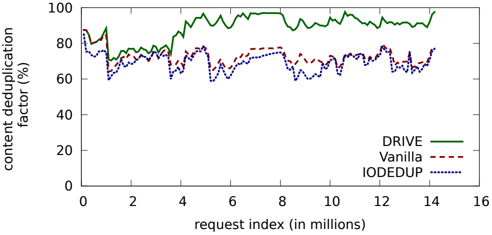


Figure: Content deduplication factor of page cache upon *webvm* trace.

Conclusions

- Both *homes* and *mail* workloads have huge number of compulsory misses, whereas the *webvm* workload has significantly fewer.
- DRIVE decreases number of capacity misses to 5% of Vanilla
- DRIVE achieves up to 97% deduplication in block-cache

Identifying similarity in multiple virtual machines

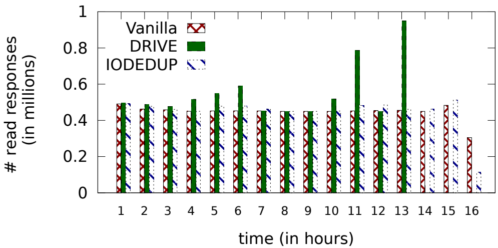


Figure: Read response throughput for aggregated (*homes+webvm*) trace.

Table: Performance for aggregated trace replay

Scheme	Cache-hit ratio (%)	Disk reads reduced(%)	Avg. read response latency (msec)
Vanilla	61.2	1.6	7.9
DRIVE	67.6	18.5	6.5
IODEDUP	62.4	4.3	7.7

Conclusions

- DRIVE completes earlier due to higher number of responses per hour on average[4].
- Huge margin in percentage of disk reads reduced

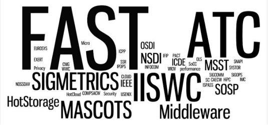
Summary of DRIVE

1. Performs implicit caching hint-based I/O redirection
2. Simulation-based evaluation shows promise—up to 97% content-deduplicated cache achieved
3. Further analysis requires more production traces

[4] Throughput derived from measured cache-hits & disk-reads and assumed latency values.

Dataset survey for I/O deduplication traces

To find for other public datasets that can be used for our evaluation



(a) Conference-name tag cloud

(b) Year-of-publication tag cloud

Three types of workloads used in literature

1. Synthetic benchmarks
2. Production I/O traces
3. Production filesystem datasets

Our requirement

I/O traces having content representation as well—not available publicly.

Literature survey for "realistic" dataset generation

Types of datasets generated

1. I/O traces (without content) [4, 5, 6, 7, 8]
2. Filesystem content (without I/O traces) [9]

Relevant characteristics for I/O traces[5]

Block accessed distribution & Jump distances—*spatial locality*
Run lengths & Block reuse distances—*temporal locality*

General approach

1. Capture Multi-dimensional distributions and/or Markov models
2. Use above captured models to create new traces with similar properties
3. Vary appropriate parameters to create different traces as necessary

[5] *webvm* and *homes* trace characterization presented in report.

Content-defined characterization of *webvm* trace

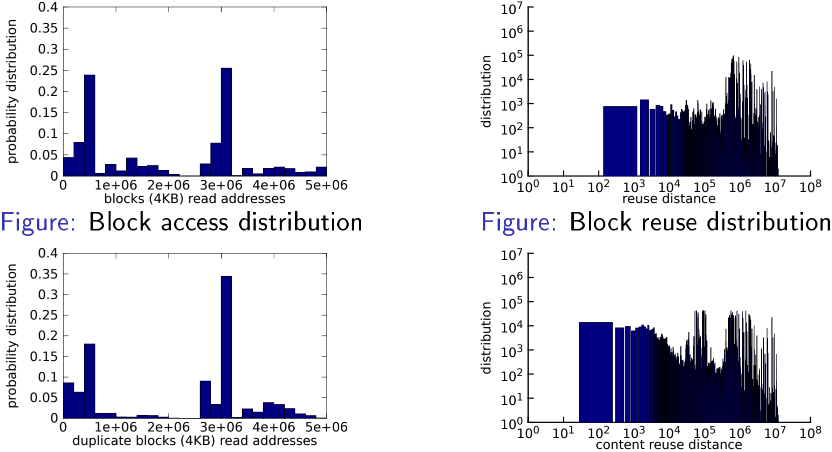


Figure: Block access distribution

Figure: Block reuse distribution

Figure: Duplicate block access distrib

Figure: Content reuse distribution

Observations

- Even duplicate content access has spatial locality property
- Temporal locality is higher for content than block

DRIVE system summary & conclusions

- In this component, we addressed I/O reduction via deduplication
- We analyzed existing work (IODEDUP) and showed that its performance is inconsistent depending on the read/write request-mix of the workload.
- We presented design & implementation of our DRIVE system
- Simulation evaluation shows promise—achieves 97% content deduplication of the host cache.
- We concluded with a survey of publicly available datasets, as well as benchmark generation literature, to make the case that future work towards I/O deduplication benchmarks is necessary

Bibliography I

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 (FAST), pages 211–224, 2010.

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.

Ricardo Koller and Raju Rangaswamy.
Trace: I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance.
Website.
http://sylab-srv.cs.fiu.edu/doku.php?id=projects:iodedup:start.

Sriram Sankar and Kushagra Vaid.
Storage Characterization for Unstructured Data in Online Services Applications.
In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC), IISWC '09, pages 148–157. IEEE Computer Society, 2009.

C. Delimitrou, S. Sankar, K. Vaid, and C. Kozyrakis.
Storage I/O Generation and Replay for Datacenter Applications.
In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 123–124, April 2011.

Christina Delimitrou, Sriram Sankar, Kushagra Vaid, and Christos Kozyrakis.
Accurate Modeling and Generation of Storage I/O for Datacenter Workloads, 2011.

Bibliography II

V. Tarasov, S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok.
Extracting Flexible, Replayable Models from Large Block Traces.
In Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST), FAST'12, pages 22–22. USENIX Association, 2012.

Zachary Kurmas, Jeremy Zito, Lucas Trevino, and Ryan Lush.
Generating a Jump Distance Based Synthetic Disk Access Pattern, 2006.

Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok.
Generating Realistic Datasets for Deduplication Analysis.
In Proceedings of the USENIX Conference on Annual Technical Conference, pages 24–24, 2012.