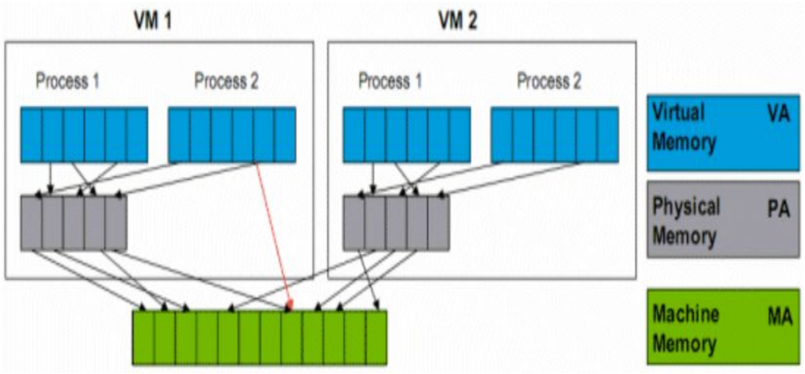
Co-operative Hypervisor Caching based on Content Similarity

Aby Sam Ross
143050093
Supervisor: Prof. Purushottam Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
11 August 2016

Memory Virtualization

- ✓ Virtualization
  - maximize utilization of resources
- ✓ Memory overcommitment
  - to collocate more VMs

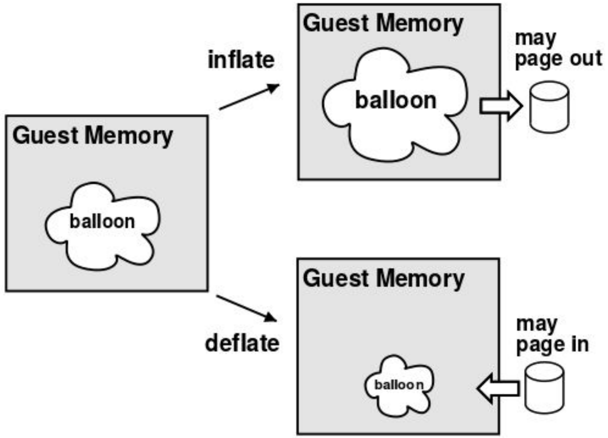


Memory Virtualization

- ✓ Memory
  - slowly “renewed”
  - useful in large granularity
  - inertia

Techniques of Memory Overcommitment

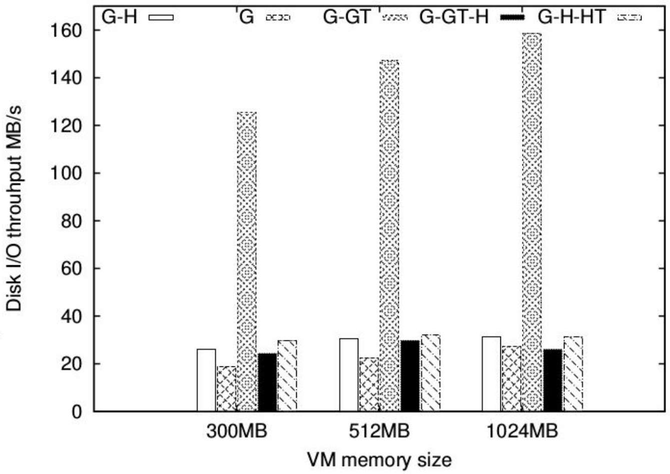
- ✓ Demand Paging by Hypervisor
- ✓ VM Migration
- ✓ Ballooning
- ✓ Content Based Page Sharing



Hypervisor caching

- ✓ a virtualized memory management technique
- ✓ memory controlled by hypervisor
  - used for the benefit of VMs without transferring ownership
  - used as optimization features that enhances VM performance
    - ➢ **second chance exclusive cache for VM page-cache that reduces disk IO**
    - ➢ in-memory swap disk for anonymous pages of a VM
  - freedom to implement custom global policies easily
    - ➢ compression, content based sharing
- ✓ can complement other techniques

Related Work [13]

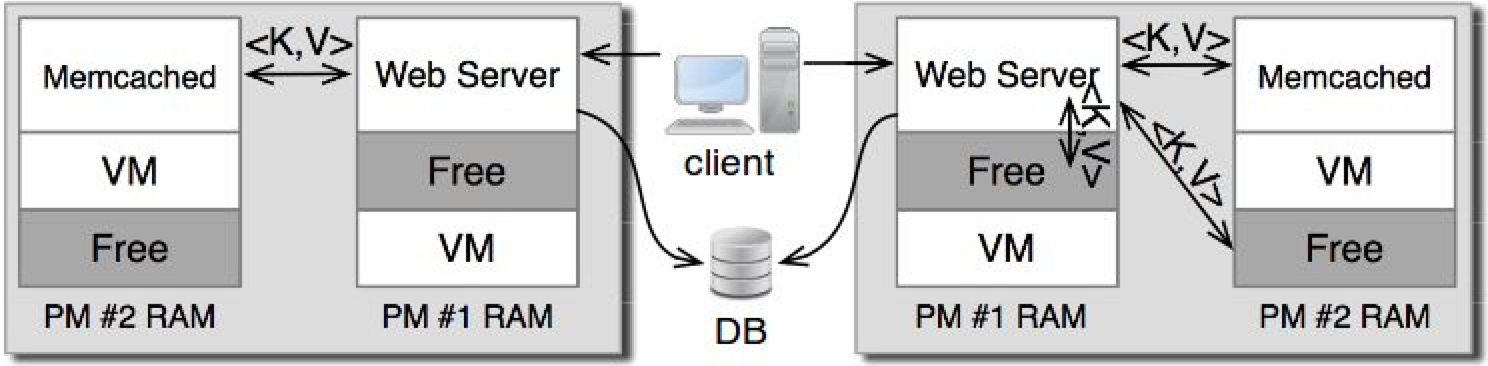


Performance metric	Performance
Disk I/O throughput	6x more
Disk read size	20x lower

Effectiveness of G-GT compared to G-H for read intensive workloads

Related Work

- ✓ 64% - 93% of all shareable pages of co-hosted VMs are page-cache pages [10]
- ✓ Mortar [7] --
  - tried make use of hypervisor caches of all PMs in a datacenter for the benefit apps in VM



Motivation

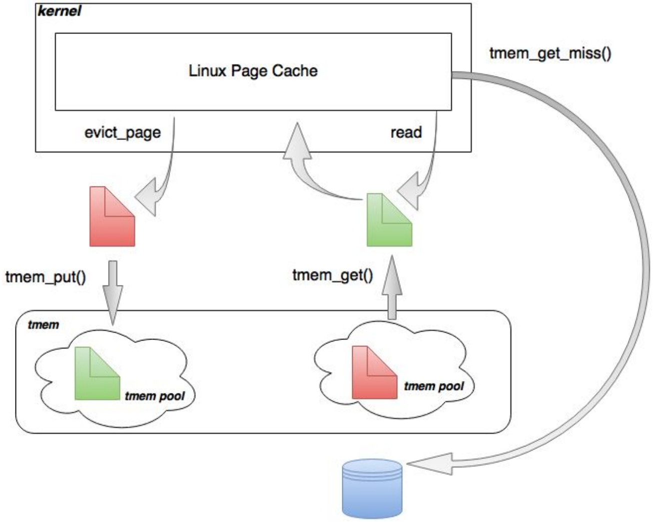
- ✓ Previous results
- ✓ Data centers
    - secondary storage accessed over network
- ✓ Presence of similar content
    - similar applications running in VM
    - common files being accessed
    - Booting VMs from same template/image

Problem Statement

- ✓ Design and implementation of a hypervisor cache, as a second chance exclusive cache for page-cache.
- ✓ Do content based page sharing to increase utilization.
- ✓ Pursue page sharing opportunities among peers in other physical machines when under memory pressure.
- ✓ Benchmark overheads of doing remote page sharing

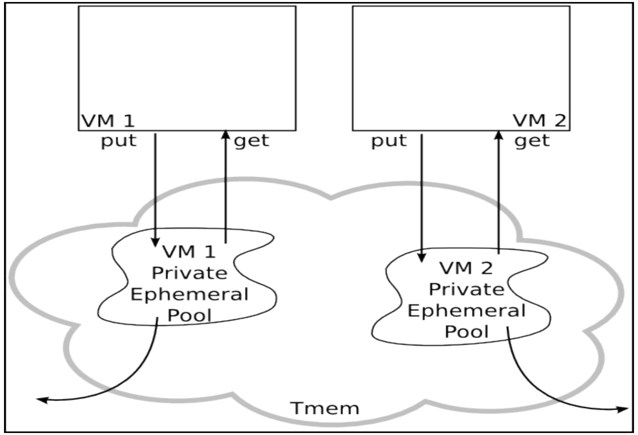
Transcendent memory

- ✓ A hypervisor caching mechanism
  - provision to be explicitly used as a second chance cache for page-cache
  - page-copy based interface to store & retrieve pages



Tmem Pools

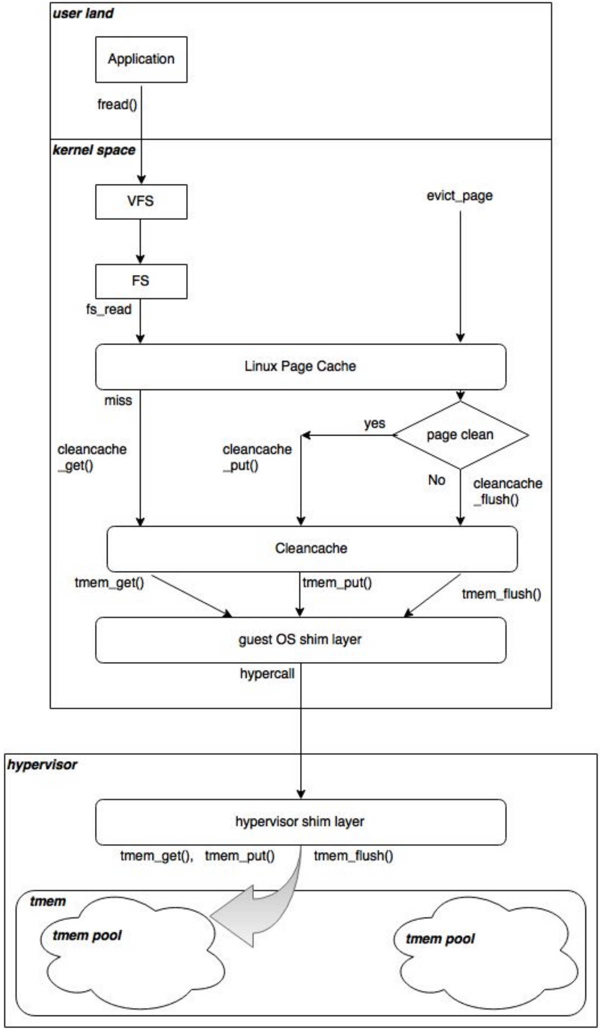
- ✓ To use tmem backend
  - VMs (clients) create pools
- ✓ Different semantics
  - Ephemeral, Persistent



flags	**Ephemeral**	**Persistent**
Private	Private to each VM. But memory can be reclaimed from this any moment. Hence pages successfully put to an ephemeral pool may or may not be present later when the client kernel uses a subsequent get with a matching handle.	Private to each VM. Pages successfully put to a persistent pool are guaranteed to be present for a subsequent get.

Tmem frontends

- ✓ source of data
  - ○ cleancache
- ✓ Tmem API
  - ○ TMEM_PUT_PAGE(poolid, objectid, pageid, pfn)
  - ○ TMEM_GET_PAGE(poolid, objectid, pageid, pfn)
- ✓ item identifier in tmem cache
  - ○ <poolid, objectid, pageid>



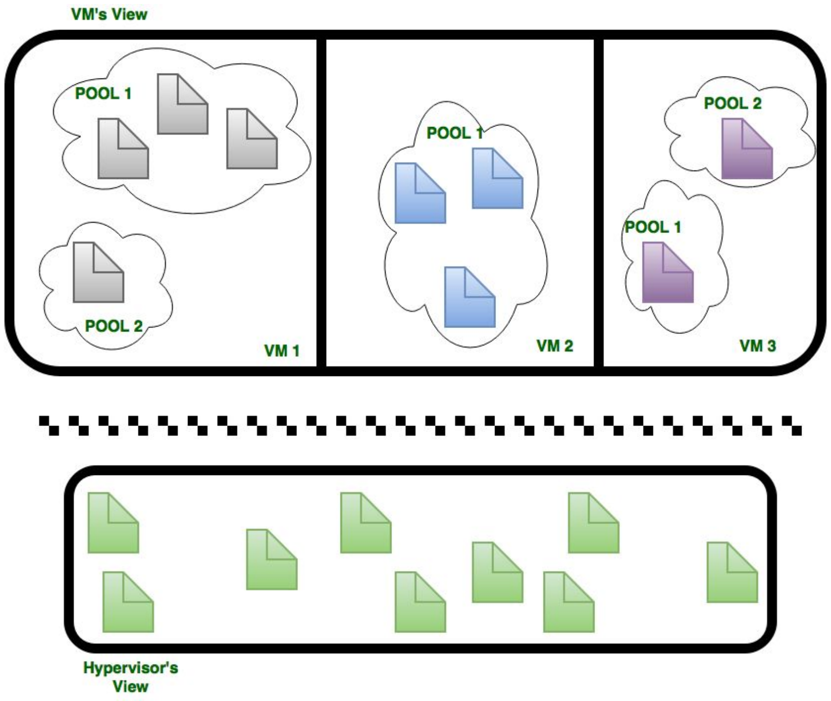
Tmem backends

✓ Different ways of implementing tmem cache
  - Xen tmem backend
  - zcache
  - RAMster
  - KVM tmem backend

Overview of related work

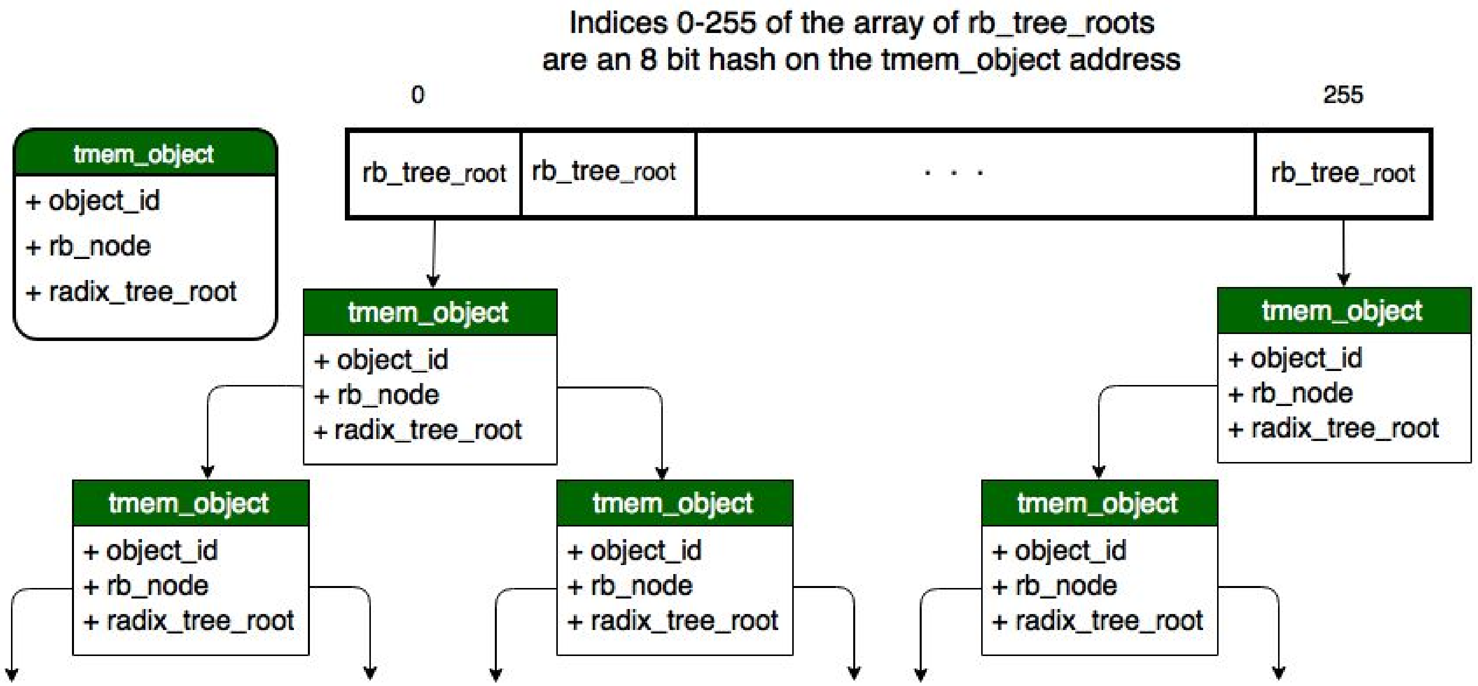
	optimizations based on content similarity		generic, application agnostic solution
	local	global	
Mortar	🚫	🚫	🚫
zcache	🚫	🚫	**Yes**
RAMster	🚫	🚫	**Yes**
Xen tmem	**Yes**	🚫	**Yes**

Design

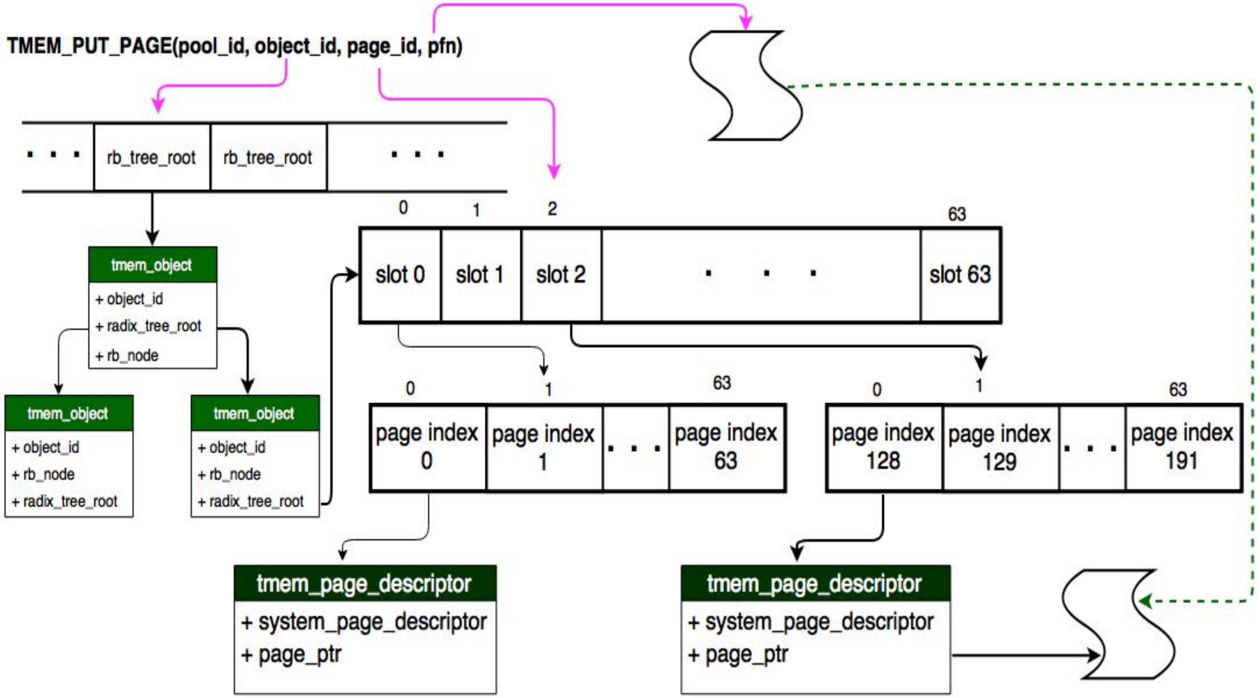


VM's View
POOL 1
POOL 2
VM 1
POOL 1
VM 2
POOL 2
POOL 1
VM 3
Hypervisor's View

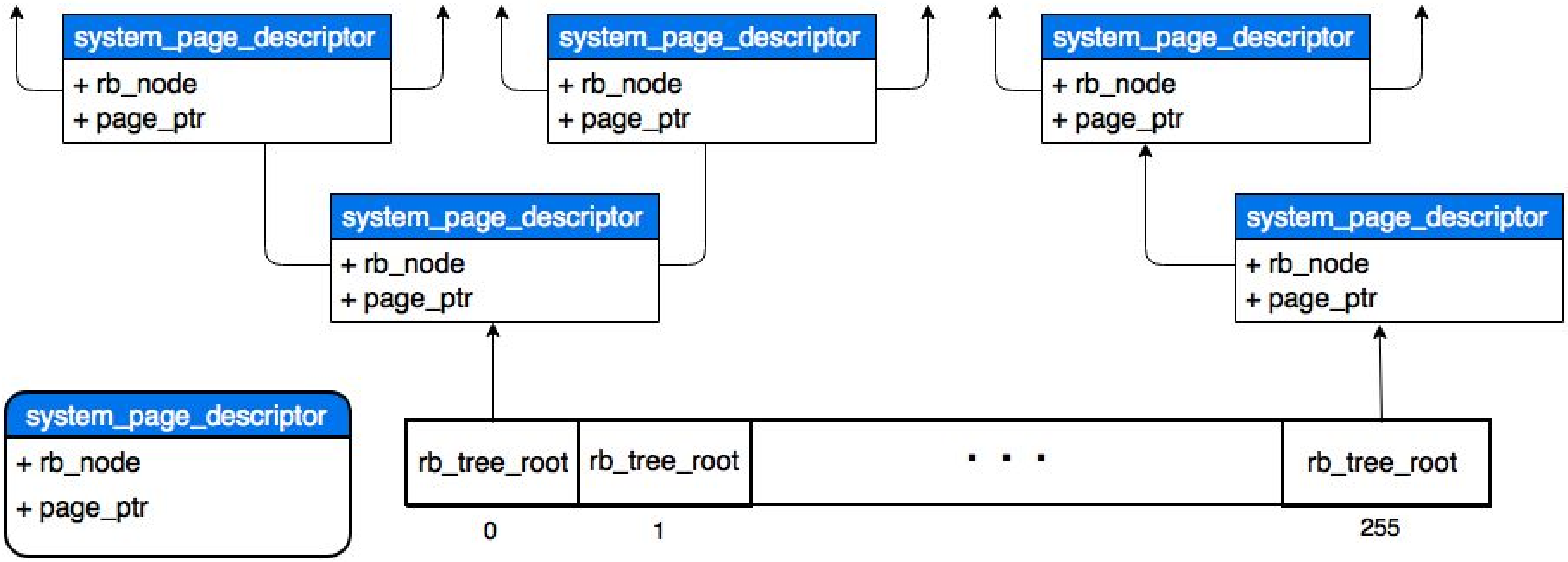
Implementation specifics - A VM tmem pool



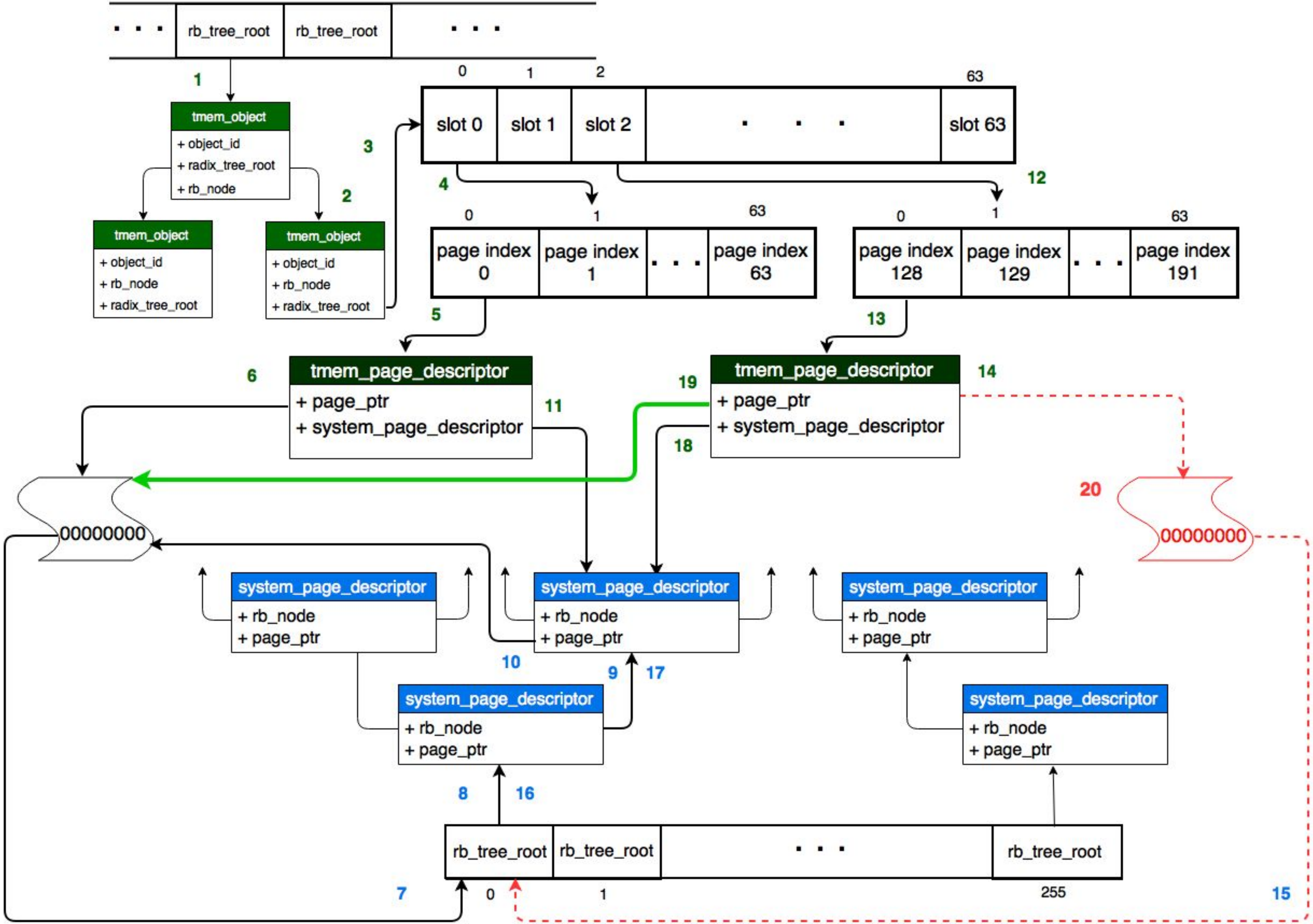
Implementation specifics - a put into a VM's pool



Implementation specifics - Hypervisor's view



The indices 0-255 of the array of
rb_tree_roots are first byte of page content



rb_tree_root
rb_tree_root
1
tmem_object
+ object_id
+ radix_tree_root
+ rb_node
2
tmem_object
+ object_id
+ rb_node
+ radix_tree_root
tmem_object
+ object_id
+ rb_node
+ radix_tree_root
3
0
1
2
63
slot 0
slot 1
slot 2
slot 63
4
12
0
1
63
page index 0
page index 1
page index 63
0
1
63
page index 128
page index 129
page index 191
5
13
6
tmem_page_descriptor
+ page_ptr
+ system_page_descriptor
11
19
tmem_page_descriptor
+ page_ptr
+ system_page_descriptor
14
18
20
00000000
00000000
system_page_descriptor
+ rb_node
+ page_ptr
system_page_descriptor
+ rb_node
+ page_ptr
system_page_descriptor
+ rb_node
+ page_ptr
10
9
17
system_page_descriptor
+ rb_node
+ page_ptr
system_page_descriptor
+ rb_node
+ page_ptr
8
16
rb_tree_root
rb_tree_root
rb_tree_root
7
0
1
255
15

Co-operative (distributed/remote) de-duplication

- ✓ Questions
  - When to do remote de-duplication?
  - Where to look?
  - What are network overheads of doing remote de-duplcation?

When to remote de-duplication?

✓ If memory is available hold on to contents
  - Avoid unnecessary network delays/overheads

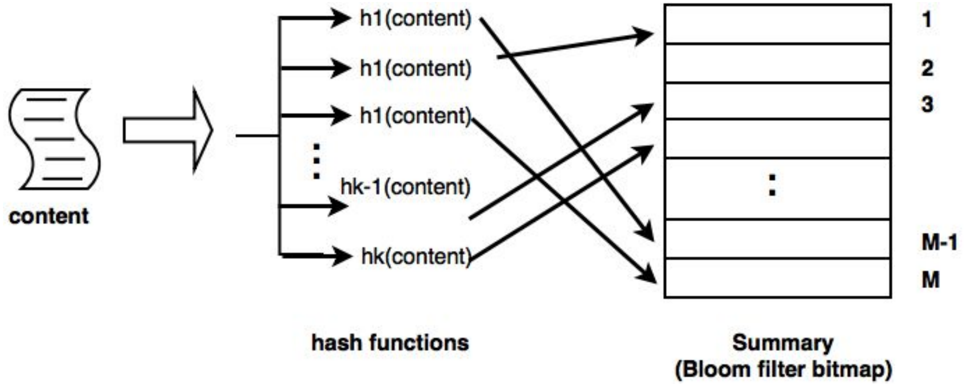
✓ Evict on memory pressure in hypervisor
  - If remote match found, well and good
  - else goes well with the ephemeral semantics of the use case (cleancache)

Where to look & overhead of this search over network.

- ✓ Ask around each co-operating backend?
- ✓ Overhead
  - H, average cache hit ratio in each host
  - N, no. of hosts is N
  - R, avg. no. of requests received
  - $(N - 1)(1 - H)R$, no. of requests each host has to handle
  - $(N)(N - 1)(1 - H)R$, requests in all
  - If N increases, there can be quadratic increase in no. of requests

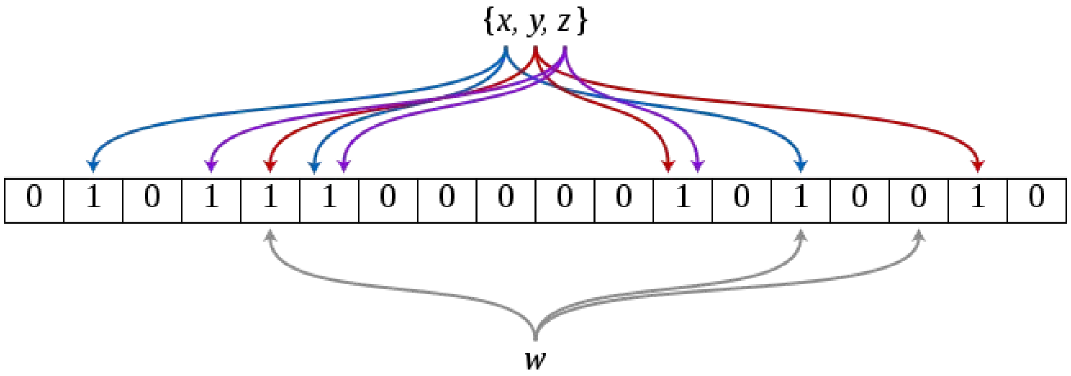
Inspiration

- ✓ Summary Cache: A web cache sharing protocol [3]
    - ○ Makes use of a memory & time efficient data structure to store contents of other proxies
        - ➢ Bloom filters
        - ➢ Avoids multicast queries among co-operating proxy caches while looking for similar content



Looking up the bloom filter

- ✓ Says that there is a probability of finding the content remotely
  - there can be false positives
    - ➢ probability of false positive (1 - (e ^ (-kn/m))) ^ K
  - free from false negatives if there are no removals



Decisions while using bloom filters

- ✓ Choosing the bitmap size
  - to reduce false positives
- ✓ Frequency of bloom filter updates
  - to reduce staleness of information
    - ➢ can cause false misses & false hits
    - ➢ false misses = missed opportunity to do remote sharing
    - ➢ false hits = incurs network overhead only to find content no longer exists

Some bloom filter stats for our use

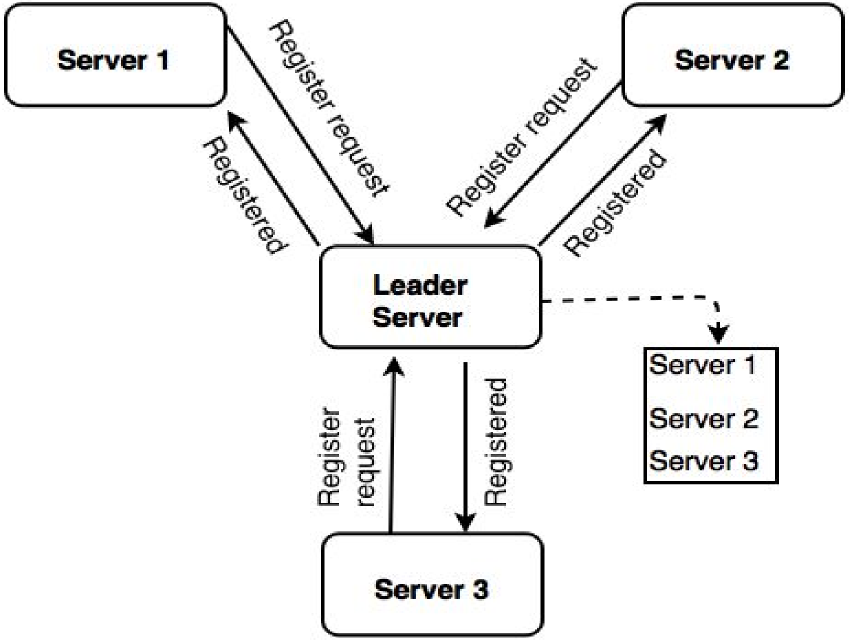
Memory of a VM (GB)	Pages	Summary pages (N)	Bit slots in bloom filter (M)	M (MB)	Total size of all bloom filters (GB)	Hash functions (K)	Probability of false positive (1-e^(-KN/M))^K
0.5	131,072	2,097,152	268,435,456	32	0.5	2	0.0002
						4	0.0000008
						8	0.0000000001
			33,554,432	4	0.0625	2	0.0138
						4	0.0024
						8	0.0006

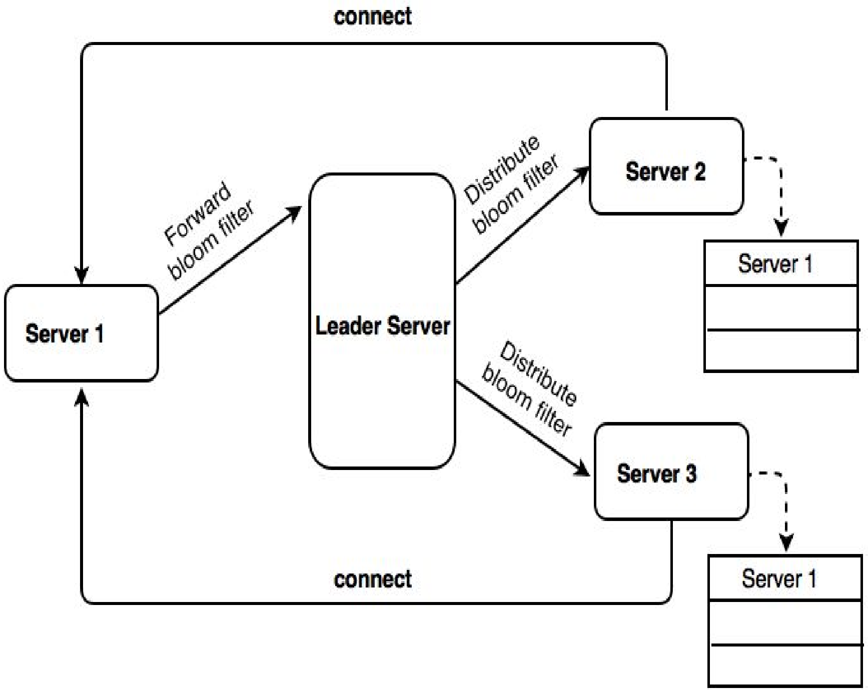
Memory of a VM (GB)	Pages	Summary pages (N)	Bit slots in bloom filter (M)	M (MB)	Total size of all bloom filters (GB)	Hash functions (K)	Probability of false positive (1-e^(-KN/M))^K
4	1,048,576	16,777,216	268,435,456	32	0.5	2	0.0138
						4	0.0024
						8	0.0006
			33,554,432	4	0.0625	2	0.3996
						4	0.5590
						8	0.8625

Bloom filter information staleness

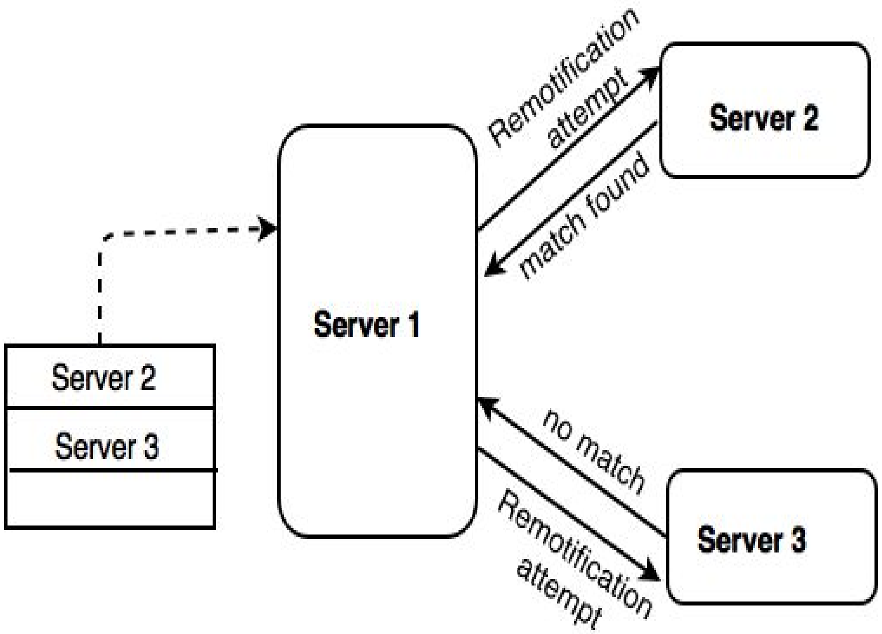
- ✓ Not a primary concern for us
  - a kernel thread exchanges bloom filters periodically with a leader server
  - a leader server takes care of forwarding bloom filters to others

Set up of the co-operative network of hypervisors

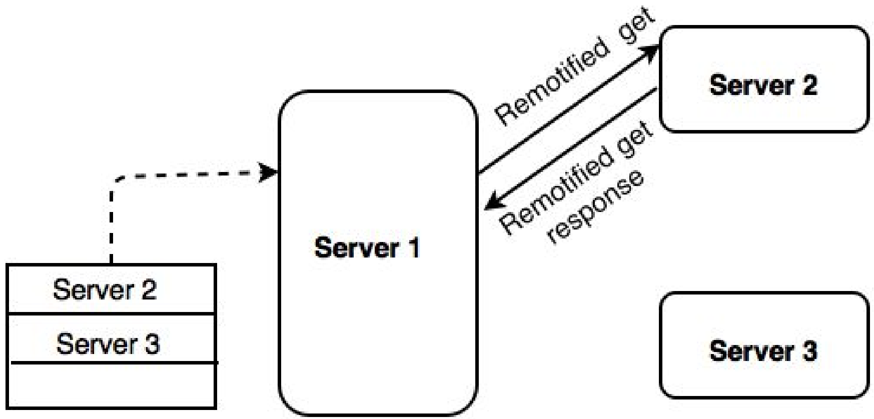




Remote de-duplication (*remotification*)



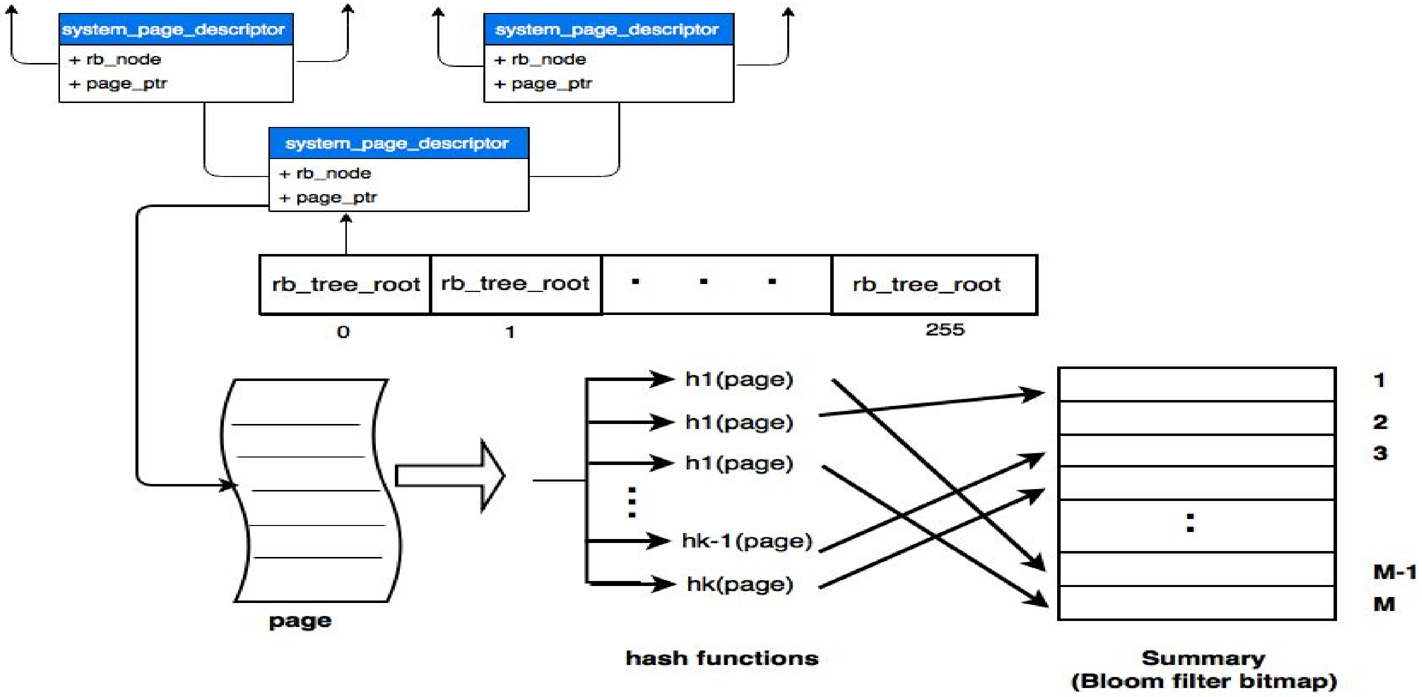
Remote retrieval (*remotified get*)



Extending local de-dup design

- ✓ Which all pages to be remotified?
    - exempt those used in local de-dup
- ✓ How to maintain state?

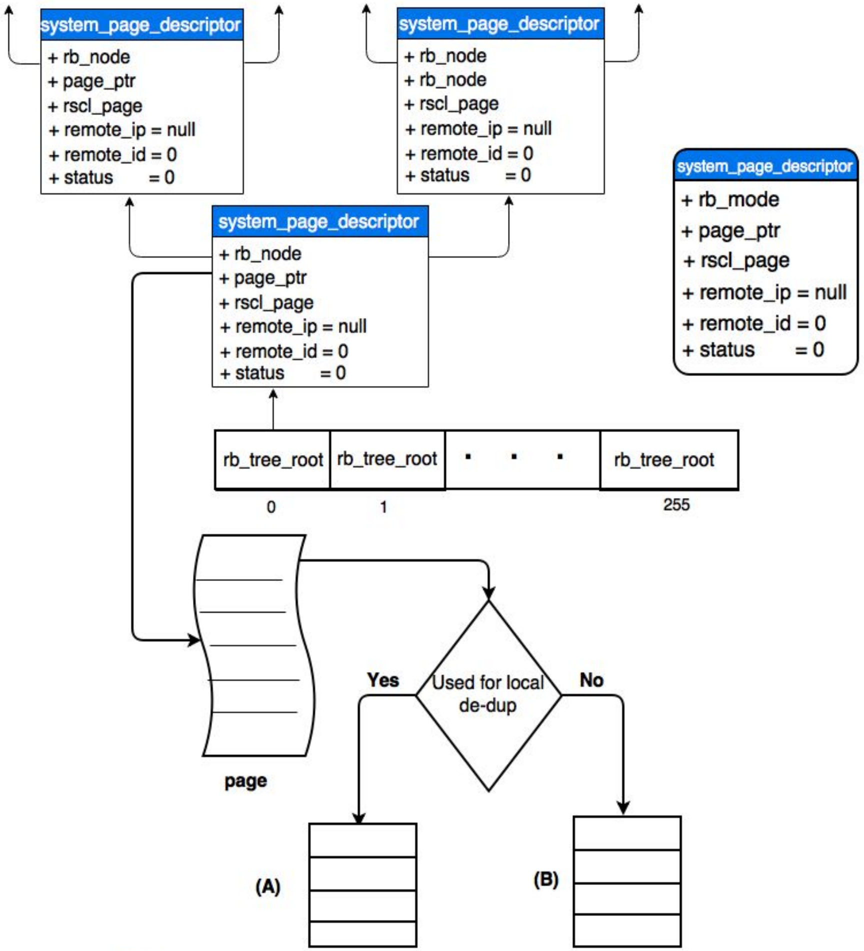
Implementation specifics - Populating the bloom filter



Maintaining state of pages held

✓ Local only list (lol)
  ○ holds pages used for local de-dup
    ➢ exempted from remote de-dup

✓ Remote sharing candidate list (rscl)
  ○ candidate pages for remote de-dup



A - Local only list

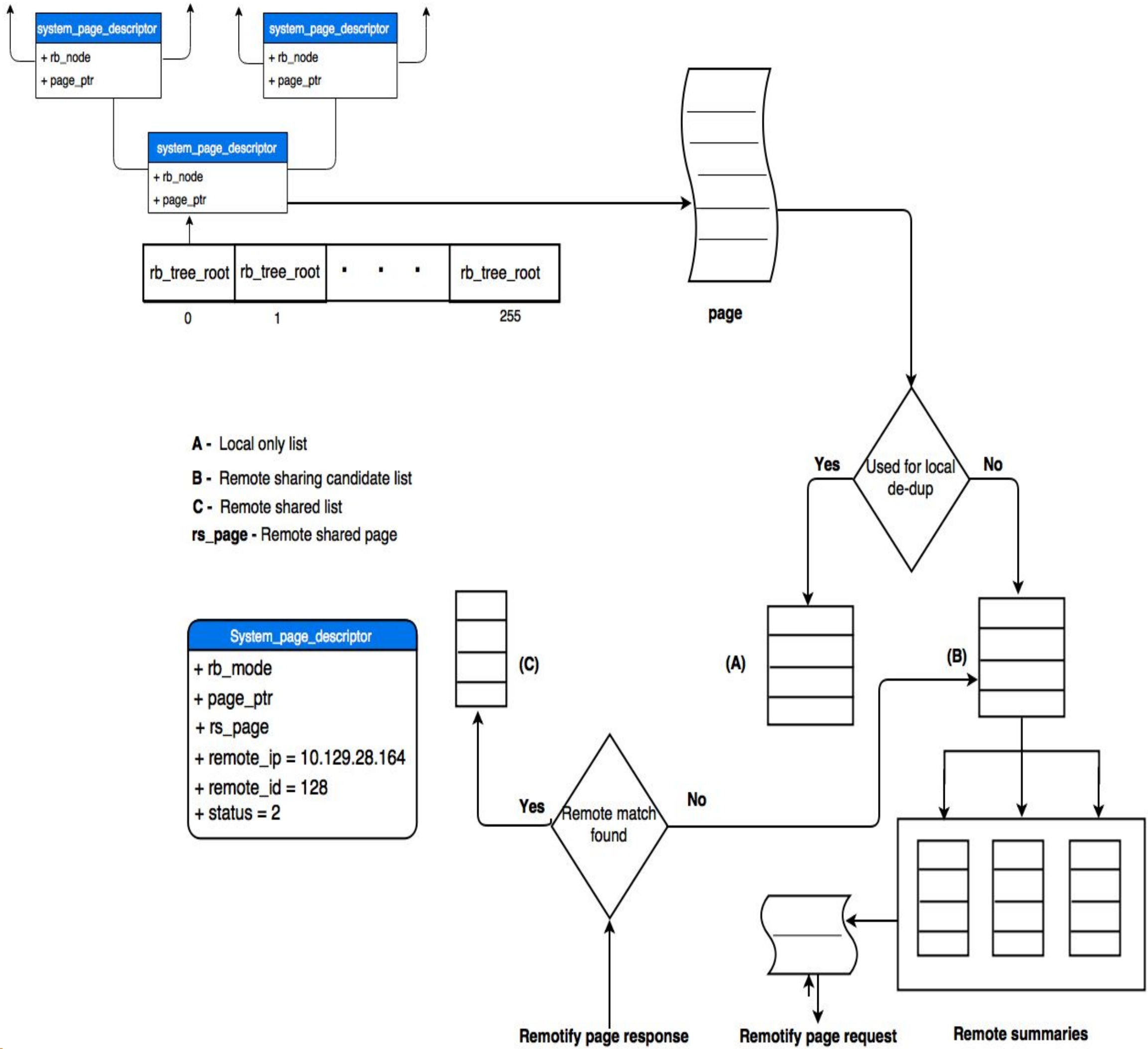
B - Remote sharing candidate list

rscl_page - Remote sharing candidate list page

Remotification on eviction

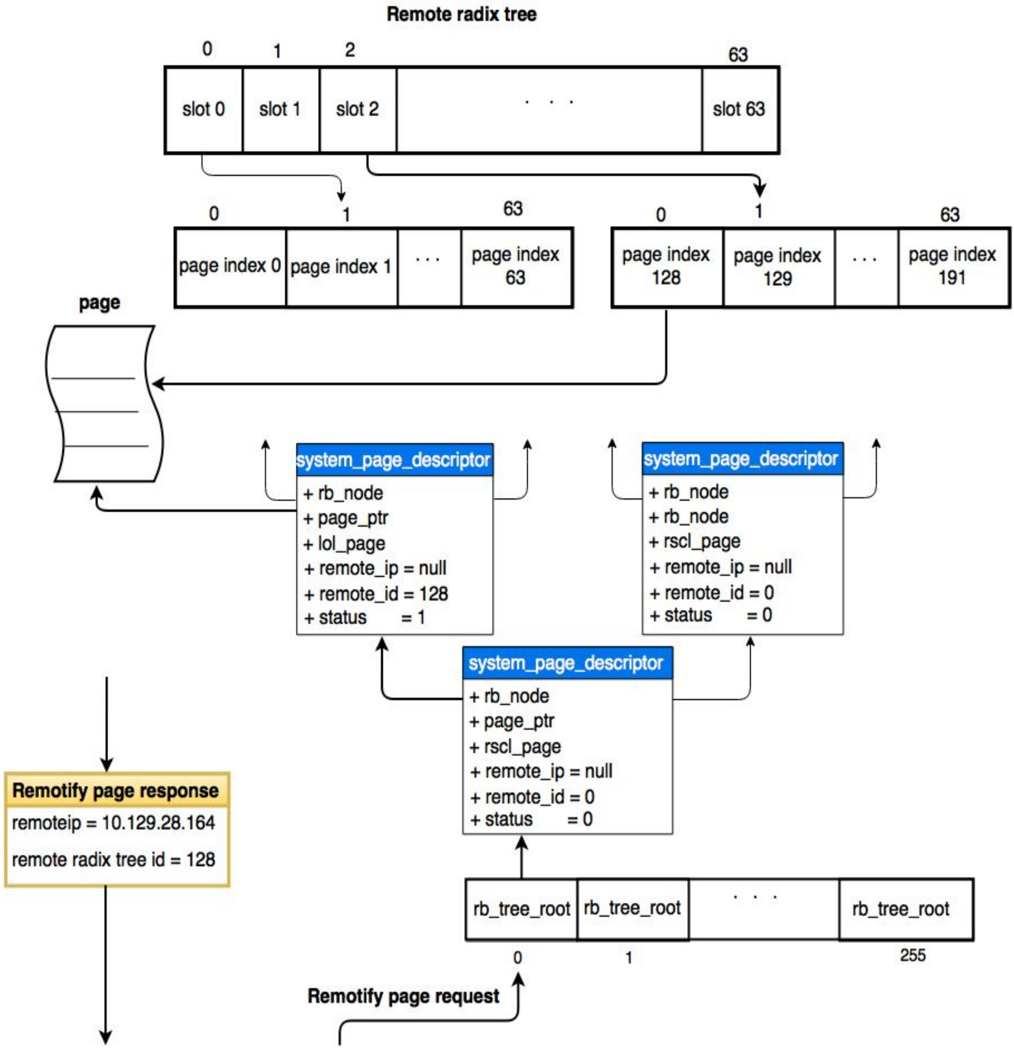
✓ Remote Shared List (rsl)

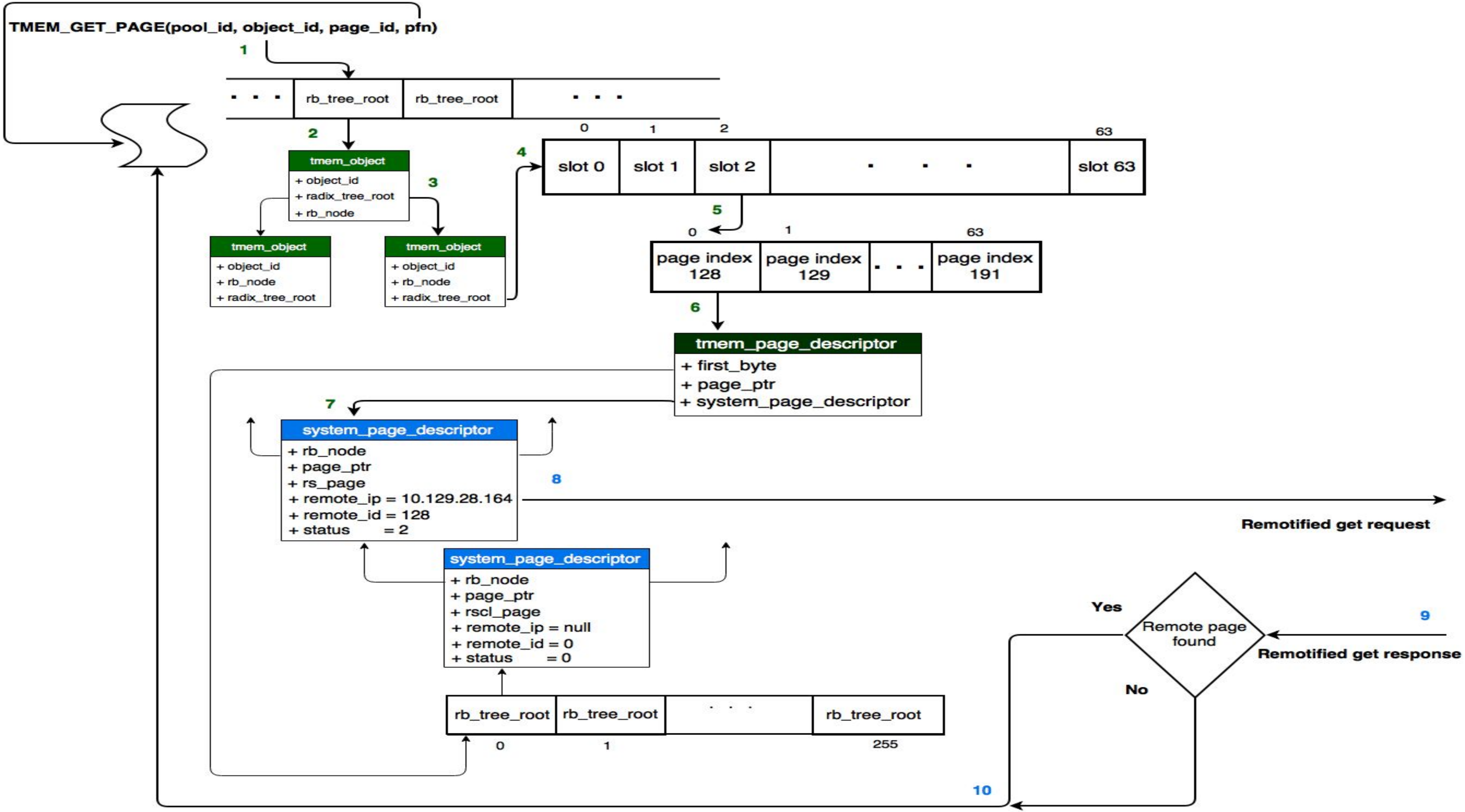
- evicted pages' reference



Handling remote de-duplication

- ✓ Remote Radix Tree
  - ○ pages used to de-dup remote page
    - ➢ returns an index within it
- ✓ moved to local only list
  - ○ prevents recursive remote de-dup





TMEM_GET_PAGE(pool_id, object_id, page_id, pfn)
1
. . .
rb_tree_root
rb_tree_root
. . .
2
tmem_object
+ object_id
+ radix_tree_root
+ rb_node
3
tmem_object
+ object_id
+ rb_node
+ radix_tree_root
tmem_object
+ object_id
+ rb_node
+ radix_tree_root
4
0 1 2 63
slot 0
slot 1
slot 2
. . .
slot 63
5
0 1 63
page index 128
page index 129
. . .
page index 191
6
tmem_page_descriptor
+ first_byte
+ page_ptr
+ system_page_descriptor
7
system_page_descriptor
+ rb_node
+ page_ptr
+ rs_page
+ remote_ip = 10.129.28.164
+ remote_id = 128
+ status = 2
8
Remotified get request
system_page_descriptor
+ rb_node
+ page_ptr
+ rscl_page
+ remote_ip = null
+ remote_id = 0
+ status = 0
Yes
Remote page found
No
9
Remotified get response
rb_tree_root
rb_tree_root
. . .
rb_tree_root
0 1 255
10

Experiments & Evaluations

- ✓ Correctness verification
  - local de-duplication
  - remote de-duplication
- ✓ Evaluation of runtime overheads
  - synthetic read intensive workload
  - local de-dup only mode
  - local & remote de-dup mode
- ✓ Micro benchmarking of tmem operations
  - same synthetic read intensive workloads
  - local & remote de-dup mode

Experiment Setup

- ✓ 3 machines.
- ✓ Machine A with 4 Intel Core i5-4440 3.10 GHz CPUs, 8 GB RAM
- ✓ Machine B with 4 Intel Core i7-3770 3.40 GHz CPUs, 8 GB RAM;
- ✓ Machine C with 4 Intel Core 2 Quad Q9550 2.83 GHz CPUs, 6 GB RAM.
- ✓ All three booted a custom Linux kernel based on linux-4.1.3.
- ✓ Machines B and C hosted 1 VM each using KVM.
- ✓ The VMs were configured to boot with 512 MB RAM and had all other configurations same.
- ✓ For remote de-dup experiments Machine A acted as the leader server

1. Correctness verification - Local de-dup

- ✓ Question
  - Whether duplicate pages are being correctly identified and the redundant copies being correctly removed from the tmem backend store?
- ✓ Experiment Setup
  - Workload
    - ➢ copying a file of 1GB full of 1s to /dev/null
    - ➢ 1 run
  - VM configurations
    - ➢ configured with 0.5 GB memory
- ✓ Observation
  - #pages that were successfully de-duplicated = 1 - #pages of file put into backend

2. Correctness verification - Remote de-dup

- ✓ Question
  - Whether matches for pages coming in from a remote backend can be found out and de-duplicated. Also check whether all such pages that were remotified be retrieved successfully. i.e. check the correct working of *remotification* and *remotified get*.
- ✓ Experiment Setup
  - Workload
    - ➢ copying a file of 1GB full of random content to /dev/null
    - ➢ 2 runs
  - 3 PMs
    - ➢ Machine A leader server, Machine B & C hosts VMs involved
    - ➢ Machine B does not evict its pages
  - 2 VMs
    - ➢ 1 each on machine B, Machine C with 0.5 GB memory

2. Correctness Verification - remote de-dup

✓ Observation

- All pages of random file that were evicted from Machine C were successfully de-duplicated at Machine B in run 1
- All pages of random file that were evicted from Machine C in run 1 were succefully retrieved from Machine C in run 2
- *Remotification* & *remotified gets* work correctly

3. Evaluation of runtime overheads - Local de-dup only mode

✓ Objective

- The overhead of a tmem backend that does local de-dup on end-end delay of a synthetic read intensive workload

✓ Experiment Setup

- Same as experiment 1
- File in VM - 1 GB rand file
- #runs 2

✓ Observation

	Cold	**Hot**
Mean Runtime	15.305	2.464
Std Deviation	0.475	0.028

Table 6.1: Effects of hot & cold tmem caches in local de-dup only mode on a synthetic workload in a VM hosted in machine B. Runtime is specified in seconds. Base case mean 14.456s. The results were obtained over 100 runs.

4. Evaluation of runtime overheads - Local & Remote de-dup mode

✓ Objective

- The overhead of a tmem backend that does both remote & local de-dup on end-end delay of a synthetic read intensive workload

✓ Experiment Setup

- Same as experiment 2

✓ Observation

	Cold	**Hot**
Mean Runtime	15.724	32.759
Std Deviation	0.688	2.194

Table 6.2: Effects of hot & cold tmem caches in local & remote de-dup mode on a synthetic workload in a VM hosted in machine B. Runtime is specified in seconds. Base case mean 14.456s. The results were obtained over 100 runs.

5. MIcro benchmarking of tmem functions - local de-dup only mode

- ✓ Objective
  - ○ Benchmark the run-time overheads of individual operations in tmem in terms of CPU cycles excluding the cycles consumed for VMEntry & VMExit
- ✓ Experiment Setup
  - ○ Same as experiment 3
  - ○ Files in VM - 1 GB rand file, 1 GB one file

5. Micro benchmarking of tmem functions - local de-dup only mode

✓ Observations

Operation	**CPU Cycles**	**Total #**
`kvm_host_get_page` **(hits)**	16143 ± 10257	263031
`kvm_host_get_page` **(miss)**	8745 ± 1951	264676
`kvm_host_put_page`	20859 ± 7278	262752
`local de-duplication` **(succ)**	0	0
`local de-duplication` **(fail)**	8138 ± 2236	262752

Table 6.3: Table depicting the overhead of tmem functions in terms of CPU cycles when the tmem cache does only local de-duplication. Test used the *rand file*. `get` miss overheads were separately calculated without populating the tmem cache.

5. Micro benchmarking of tmem functions - local de-dup only mode

✓ Observations

i

Operation	CPU Cycles	Total #
`kvm_host_get_page` **(hits)**	7440 ± 6037	263166
`kvm_host_get_page` **(miss)**	9014 ± 2026	278397
`kvm_host_put_page`	23655 ± 9137	262753
`local de-duplication` **(succ)**	10048 ± 2825	261381
`local de-duplication` **(fail)**	8325 ± 4098	1372

Table 6.4: Table depicting the overhead of tmem functions in terms of CPU cycles when the tmem cache does only local de-duplication. Test used the *one file*. `get` miss overheads were separately calculated without populating the tmem cache.

6. MIcro benchmarking of tmem functions - local & remote de-dup mode

- ✓ Objective
  - Benchmark the run-time overheads of individual operations in tmem in terms of CPU cycles excluding the cycles consumed for VMEntry & VMExit
- ✓ Experiment Setup
  - Same as experiment 4

6. Micro benchmarking of tmem functions - local & remote de-dup mode

✓ Observations

Operation	CPU Cycles	Total #
kvm_host_get_page (hit)	379232 ± 387284	258503
kvm_host_get_page (miss)	0	0
kvm_host_put_page	107007 ± 241533	258036
remotify_page (succ)	960450 ± 76539	155316
remotify_page (fail)	0	0
remotified_get_page (succ)	623392 ± 305216	152256
remotified_get_page (fail)	0	0

Table 6.5: Table depicting the overhead of tmem functions in terms of CPU cycles when the tmem cache does both local & remote de-duplication.

6. Micro benchmarking of tmem functions - local & remote de-dup mode

✓ Observations

Operation	time in ms
kvm_host_get_page (hit)	0.77
kvm_host_get_page (miss)	0
kvm_host_put_page	0.34
remotify_page (succ)	1
remotify_page (fail)	0
remotified_get_page (succ)	0.92
remotified_get_page (fail)	0

Table 6.6: Table depicting the max possible overhead of tmem functions in terms of time in ms when tmem cache does both local & remote de-duplication. The frequency of CPU is assumed to be 1GHz

6. Micro benchmarking of tmem functions - local & remote de-dup mode

✓ Comparing with std values of network disk access [11]

Parameters	Value
Model	Seagate 39102FC
Interface	Fibre Channel
Capacity	9.1 Gbytes
Cache size	1 Mbyte
Rotational speed	10,025 RPM
Avg. rotational time	3.0 ms
Seek time	Read 5.4 ms, write 6.2 ms
Internal transfer rate	19.0–28.4 MB/s

Table 1. *Disk parameters.*

	Read (ms)			
Configuration	1 kB	4 kB	16 kB	64 kB
SAN	9.04	9.19	10.21	14.38
GigE LAN iSCSI	9.57	9.7	10.91	15.82
Campus LAN iSCSI	11.37	12.45	14.33	24.2

Table 2. *Latency under light load; kB: kilobytes.*

6. Micro benchmarking of tmem functions - local & remote de-dup mode

✓ Comparing with std values of network disk access

Data size	1 kB (ms)		4 kB (ms)		16 kB (ms)		64 kB (ms)		256 k (ms)	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
iSCSI read	18.5	27.1	20.5	25.4	34.8	9.0	49.1	10.5	93.5	14.6
SMB read	19.1	28.4	22.1	25.1	40.0	8.8	62.3	16	125.1	19.4

Data size	1 kB (ms)		4 kB (ms)		16 kB (ms)		64 kB (ms)		256 k (ms)	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
iSCSI read	4.36	3.6	4.43	4.4	7.13	3.7	19.89	3.5	61.04	5.3
NFS read	9.00	2.4	9.80	2.1	12.07	2.2	25.63	2.8	75.34	9.3

Conclusions & Future Work

- ✓ Conclusions
  - Implemented a tmem backend that does local & remote de-duplication
  - First steps towards benchmarking overheads associated with remote operations
- ✓ Future work
  - compare overheads of VMEntry & VMExit with tmem operations
  - tests to be carried out with realistic workloads

References

1. lwn: cleancache, 2010, “[ocfs2-devel] [patch v3 0/8] cleancache: overview,” [Online; accessed 18-October-2015], https://oss.oracle.com/pipermail/ocfs2-devel/2010-June/006723.html
2. lwn: in kernel compression, 2013, “In-kernel memory compression [lwn.net],” [Online; accessed 18-October-2015], http://lwn.net/Articles/545244/
3. Fan, Li, Pei Cao, Jussara Almeida, and Andrei Z Broder, 2000, “Summary cache: a scalable wide-area web cache sharing protocol,” IEEE/ACM Transactions on Networking (TON) 8, 281–293
4. Filters, Bill Mill: Bloom, 2010, “Bloom filters by example,” [Online; accessed 18-October-2015], http://billmill.org/bloomfilter-tutorial/
5. lwn: tmem frontends, 2010, “Cleancache and frontswap [lwn.net],” [Online; accessed 18-October-2015], https://lwn.net/Articles/386090/
6. frontswap, Kernel.org:, 2010, “Kernel documentation:Frontswap,” [Online; accessed 18-October-2015], https://www.kernel.org/doc/Documentation/vm/frontswap.txt

References

7. Hwang, Jinho, Ahsen J Uppal, Timothy Wood, and H Howie Huang, 2013, “Mortar:filling the gaps in data center memory,” in Proceedings of the 4th annual Symposium on Cloud Computing (ACM) p. 30
8. Magenheimer, Dan, Chris Mason, Dave McCracken, and Kurt Hackel, 2009, “Transcendent memory and linux,” in Proceedings of the Linux Symposium, pp. 191–200
9. lwn: tmem in a nutshell, 2010, “Transcendent memory in a nutshell [lwn.net],” [Online;accessed 18-October-2015], https://lwn.net/Articles/454795/
10. Kloster, Jacob Faber, Jesper Kristensen, and Arne Mejlholm, 2007, “Determining the use of interdomain shareable pages using kernel introspection,” Department of Computer Science, Aalborg University
11. Lu, Yingping, and David HC Du, 2003, “Performance study of iscsi-based storage sub-systems,” IEEE communications magazine 41, 76–82

References

13. Mishra, Debadatta, and Purushottam Kulkarni, 2014, “Comparative analysis of page cache provisioning in virtualized environments,” in Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on (IEEE) pp. 213–222