CMD: Classification-based Memory Deduplication through Page Access Characteristics

Licheng Chen, Zhipeng Wei, Zehan Cui, Mingyu Chen, Haiyang Pan, and Yungang Bao

Institute of Computing Technology (ICT)

Chinese Academy of Sciences (CAS)

VEE 2014

March 1-2, 2014

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

Background

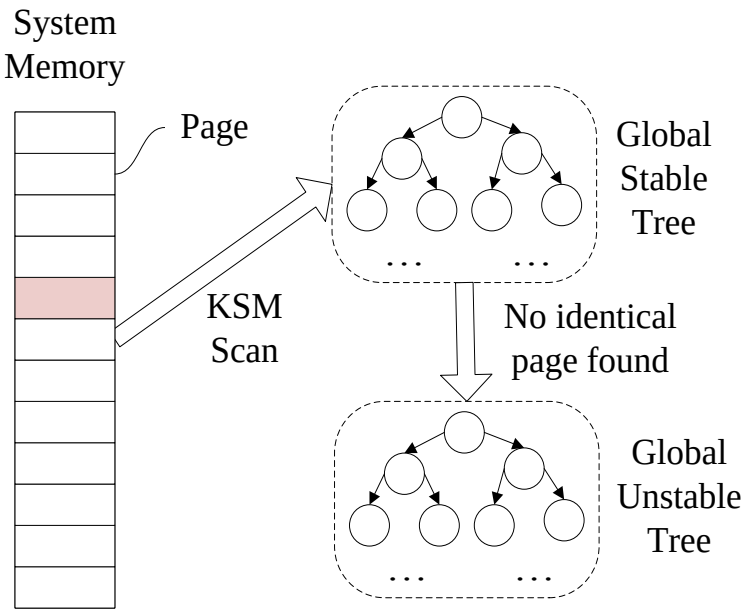
- Limited main memory size is the major bottleneck to consolidate more virtual machines on a hosting server.
  - Increasing number of cores integrated into processor
  - Larger working set of workloads running in VMs.

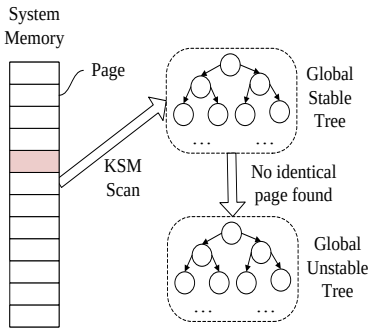
Background

- Limited main memory size is the major bottleneck to consolidate more virtual machines on a hosting server.
  - Increasing number of cores integrated into processor
  - Larger working set of workloads running in VMs.
- Content Based Page Sharing (CBPS) is an efficient memory deduplication technique
  - Perform page scan transparently in the hypervisor layer
  - Identical pages (with same content) are detected and shared into a single copy
  - KSM: A widely used implementation of CBPS

KSM

- Kernel Samepage Merging
  - Integrated into Linux kernel archive since 2.6.32
  - The whole memory pages are maintained into two global comparison trees
    - Stable tree: already shared pages with COW protection
    - Unstable tree: pages that are not shared





Problems with KSM

- Pages are directly compared with content (e.g. memcmp in Linux): CPU overhead
- Futile Comparison:
  - Page comparisons that fail to find any page with the same content (including the stable tree and unstable tree)
  - Pages are compared with a large number of uncorrelated pages in the global trees

KSM

- Two parameters to control KSM performance
  - Pages_to_scan: the number of pages to be scanned before sleep, it is 100 by default
  - Sleep_millisecs: the time to sleep (in milliseconds), it is 20 by default

KSM

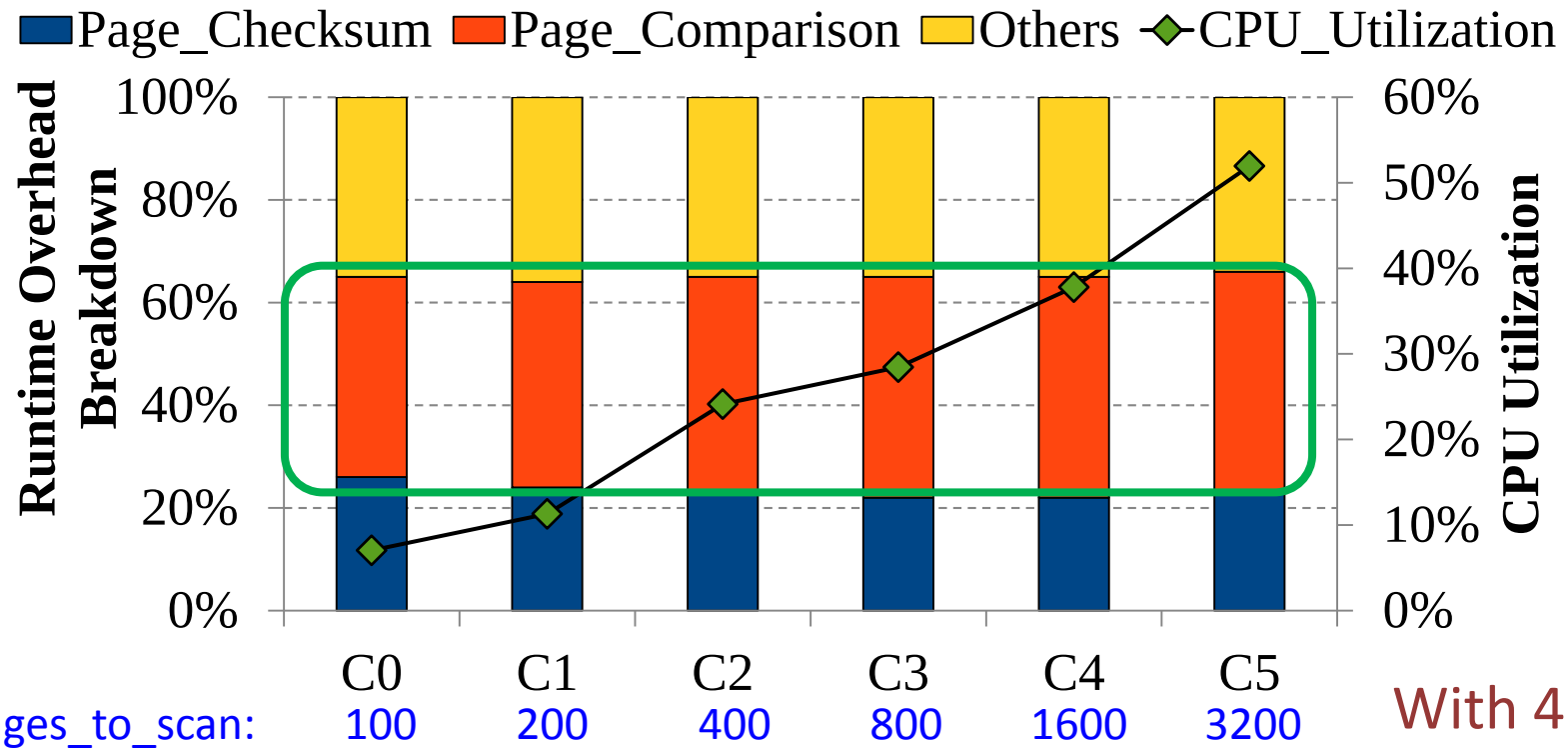
- Two parameters to control KSM performance
  - Pages_to_scan: the number of pages to be scanned before sleep, it is 100 by default
  - Sleep_millisecs: the time to sleep (in milliseconds), it is 20 by default
- KSM run-time overhead breakdown
  - Page Comparison: page content comparison in both global stable tree and unstable tree.
  - Page Checksum: calculating page checksum to determine whether a page is volatile
  - Others: other overhead, such as inserting pages in the tree, break COW when a shared page is written

Outline

- Background & Motivation
  - The profiling of KSM
- CMD: **C**lassification based **M**emory **D**eduplication
- Experimental Results
- Conclusion

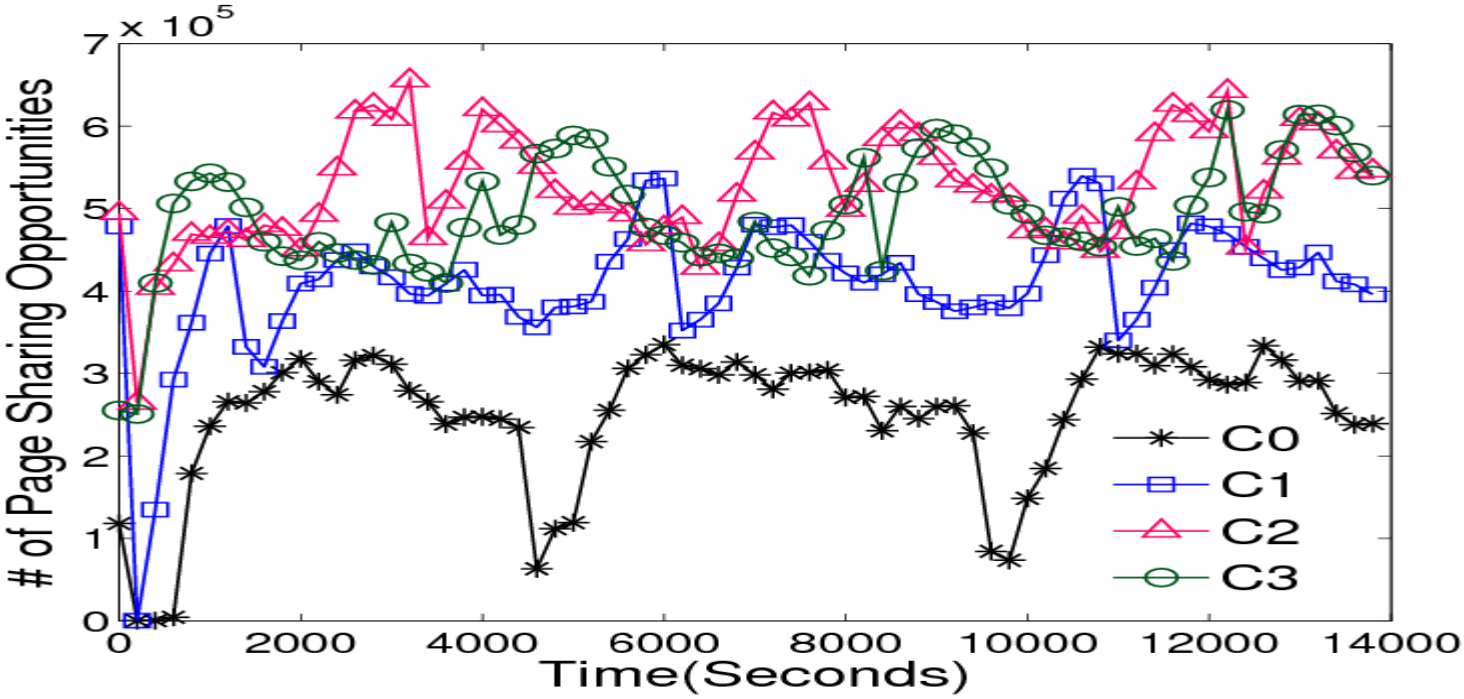
Motivation: Profiling of KSM

- The page_comparison contributes about 44% of the overall run-time overhead
- The CPU Utilization increases as more frequent page comparisons:
  - It is about 7% for C0, 24% for C2 and up to 52% for C5



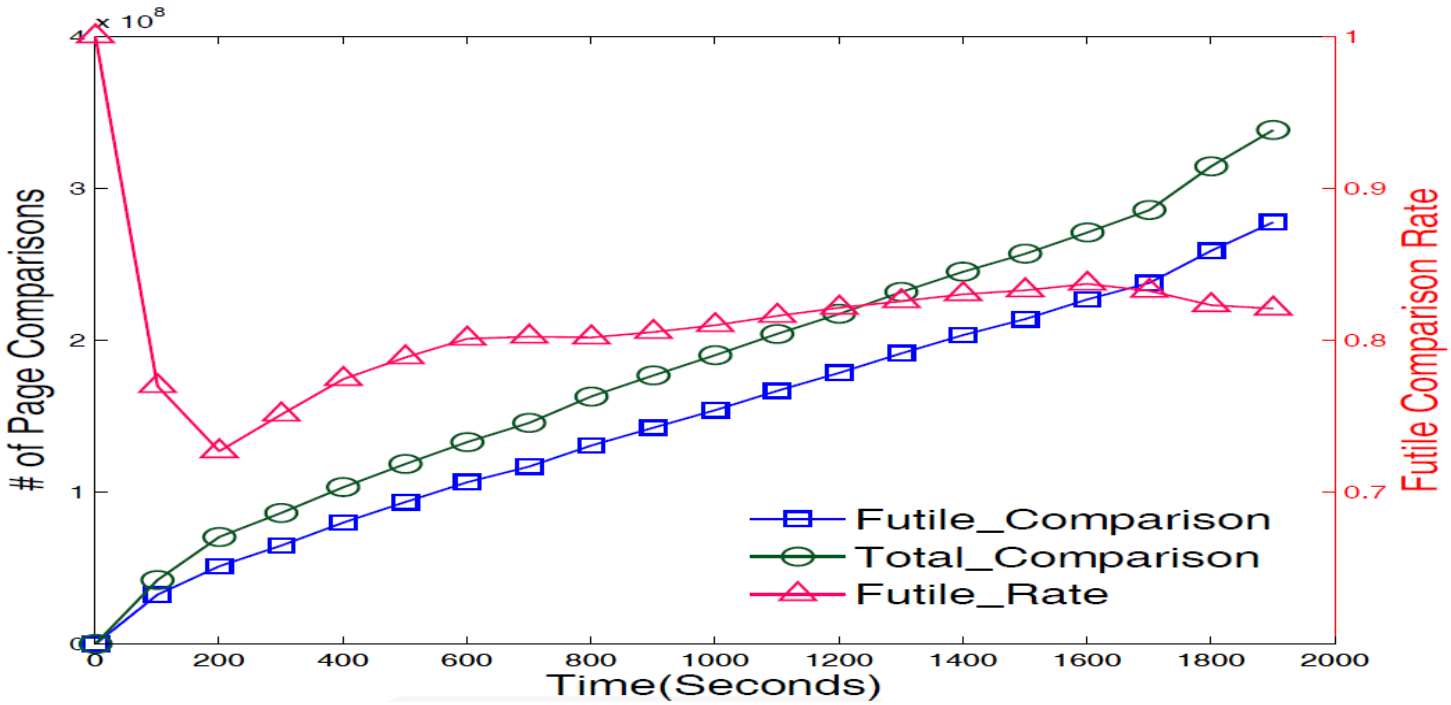
Motivation

- With more frequent page comparisons, the KSM can detect more page sharing opportunities
  - Detect more short-lived page sharing quickly



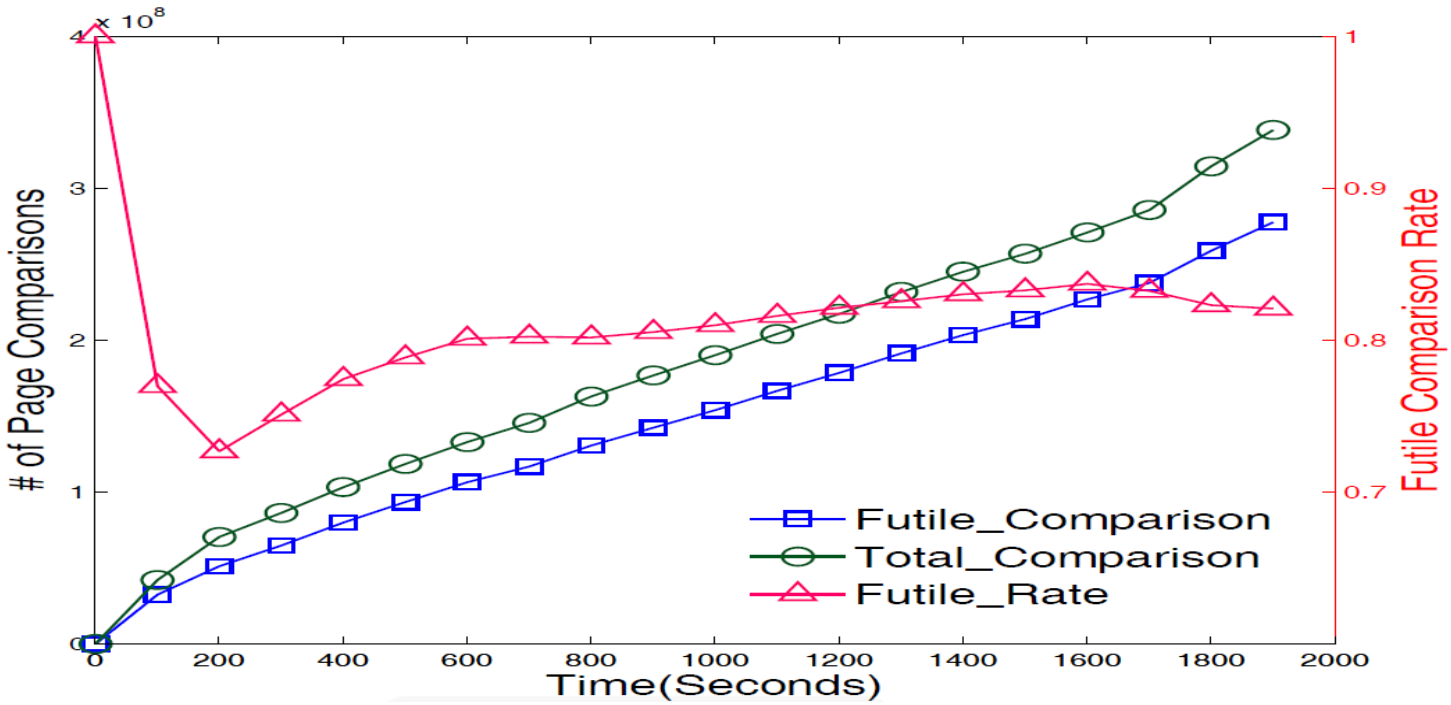
Motivation

- The total page comparison and futile comparison increase proportionally as the KSM scans periodically



Motivation

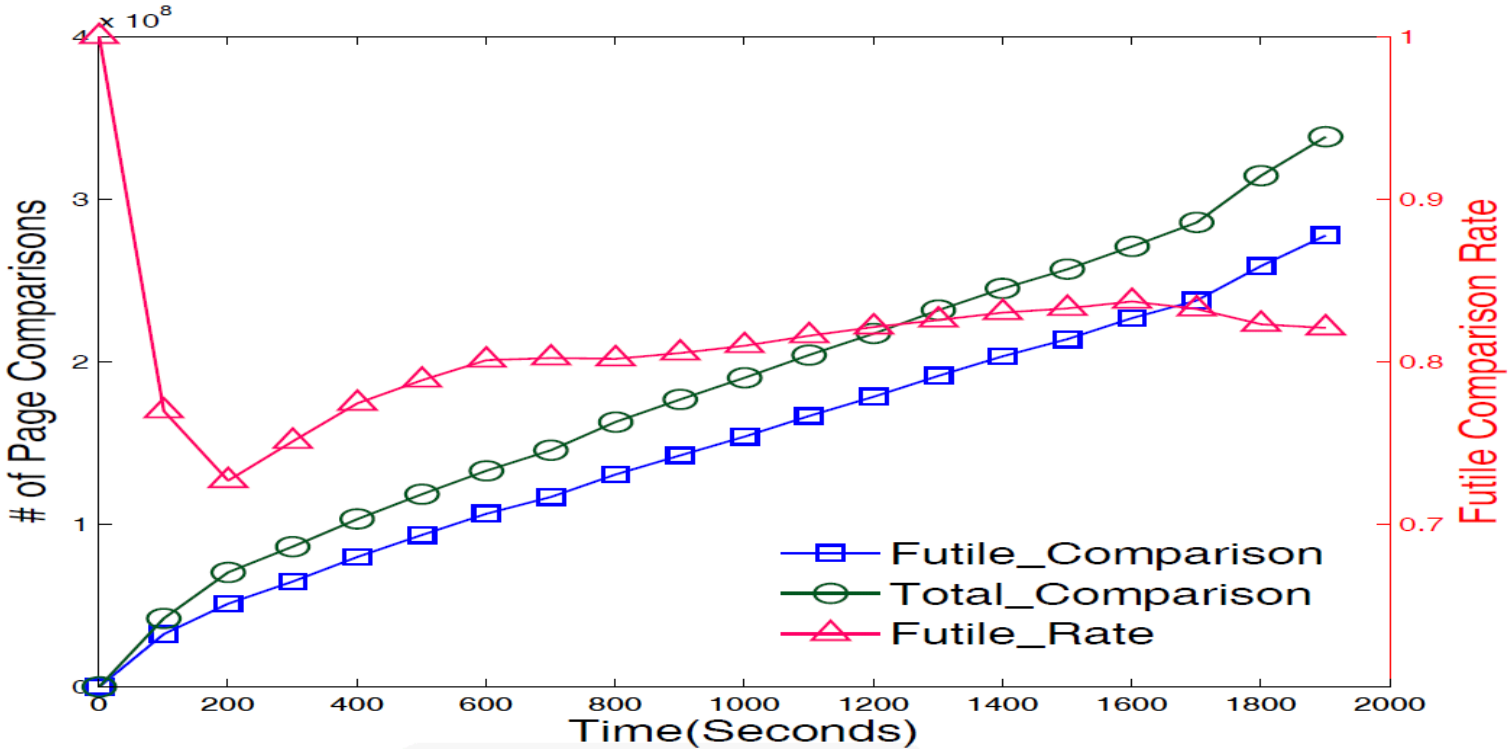
- The total page comparison and futile comparison increase proportionally as the KSM scans periodically
- Futile_Rate: the ratio between Futile_Comparison and Total_Comparison
  - It becomes steady at about 83%



Motivation

- The total page comparison and futile comparison increase proportionally as the KSM scans periodically

More frequent scan can detect more page sharing opportunities, but it also results in a large number of futile page comparisons and thus heavy CPU overhead

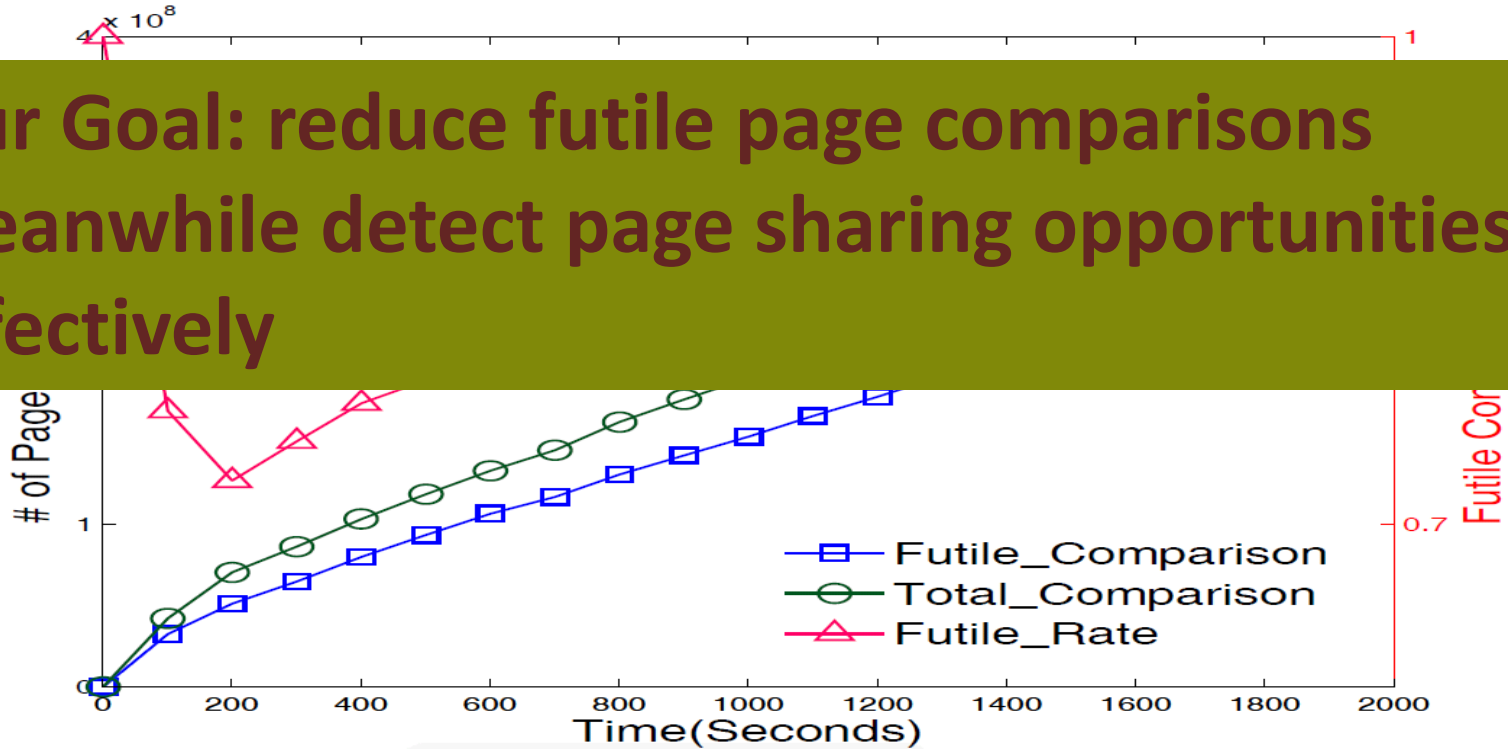


Motivation

- The total page comparison and futile comparison increase proportionally as the KSM scans periodically

More frequent scan can detect more page sharing opportunities, but it also results in a large number of futile page comparisons and thus heavy CPU overhead

Our Goal: reduce futile page comparisons meanwhile detect page sharing opportunities effectively



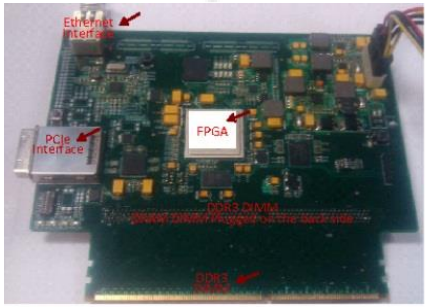
Outline

- Background & Motivation
  - The profiling of KSM
- CMD: **C**lassification based **M**emory **D**eduplication
- Experimental Results
- Conclusion

The Overview of CMD

Physical Address Trace

0x398f24a, r
0x398f24b, r
0x398f24c, w
……
0x1af4aa, w
0x1af4a6, r
0x1af4a8, w
……
0x38d2cfc, r
0x38d2cfd, w
……



Page Access Monitor

The Overview of CMD

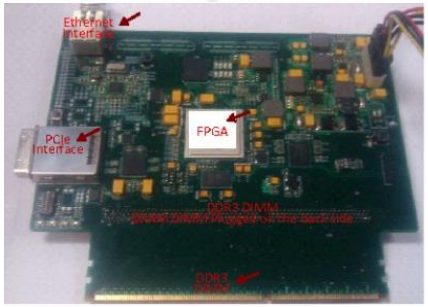
Physical Address Trace

0x398f24a, r
0x398f24b, r
0x398f24c, w
……
0x1af4aa, w
0x1af4a6, r
0x1af4a8, w
……
0x38d2cfc, r
0x38d2cfd, w
……

pfn

Page Access Characteristics

0	12	17	25	8
1	31	28	2	11
2	3	0	53	1
	……			
N-2	0	0	0	0
N-1	36	0	0	1



Page Access Monitor

The Overview of CMD

Physical Address Trace

0x398f24a, r
0x398f24b, r
0x398f24c, w
……
0x1af4aa, w
0x1af4a6, r
0x1af4a8, w
……
0x38d2cfc, r
0x38d2cfd, w
……

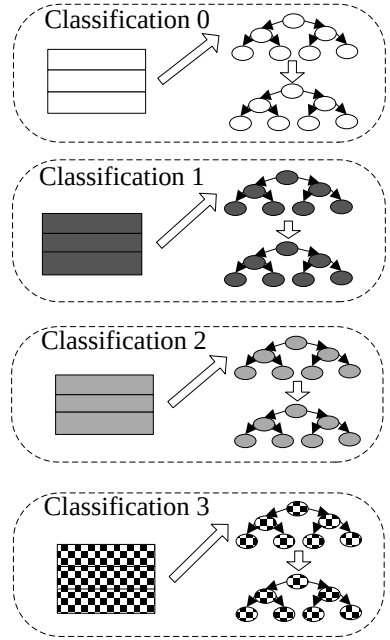
pfn

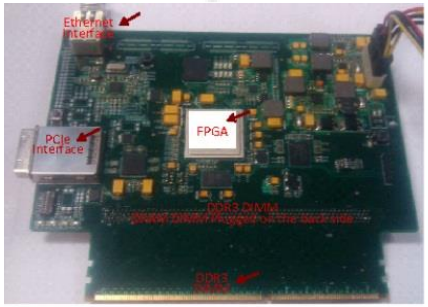
Page Access Characteristics

0	12	17	25	8
1	31	28	2	11
2	3	0	53	1
	……			
N-2	0	0	0	0
N-1	36	0	0	1

Page Classification

Classification based KSM





Page Access Monitor

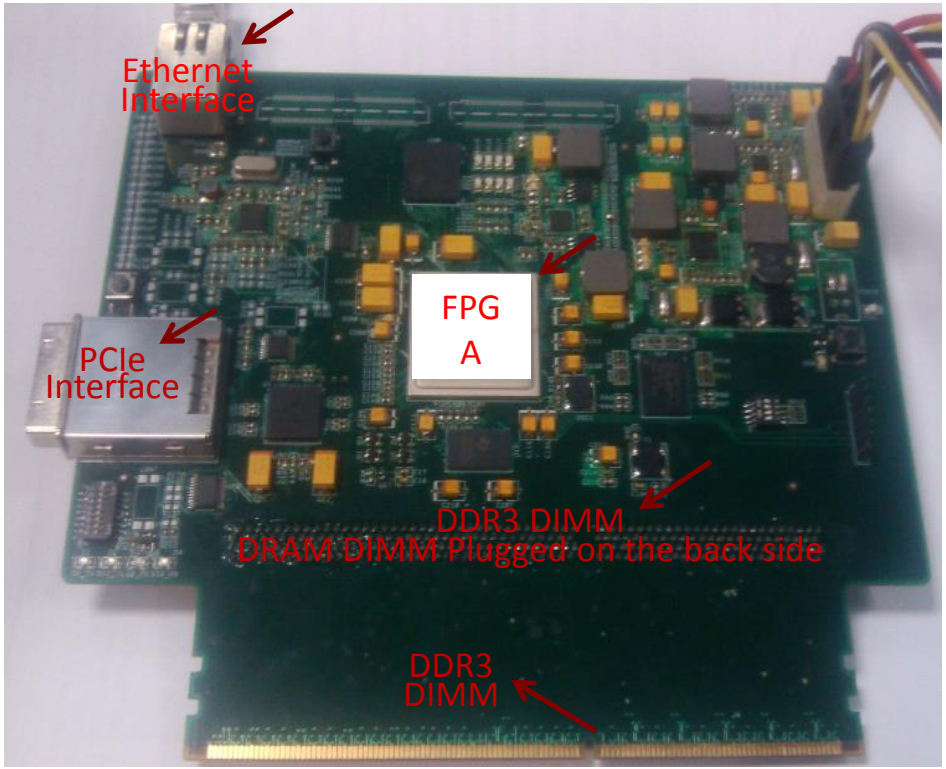
Linux

Page Access Monitor

- HMTT: Hybrid Memory Trace Toolkit
  - A DDR3 SDRAM compatible memory monitoring system
  - Adopts hardware snooping technology

Memory Trace:

- Fine granularity: cache block
- <time_stamp, r/w, phy_addr>



Page Access Monitor

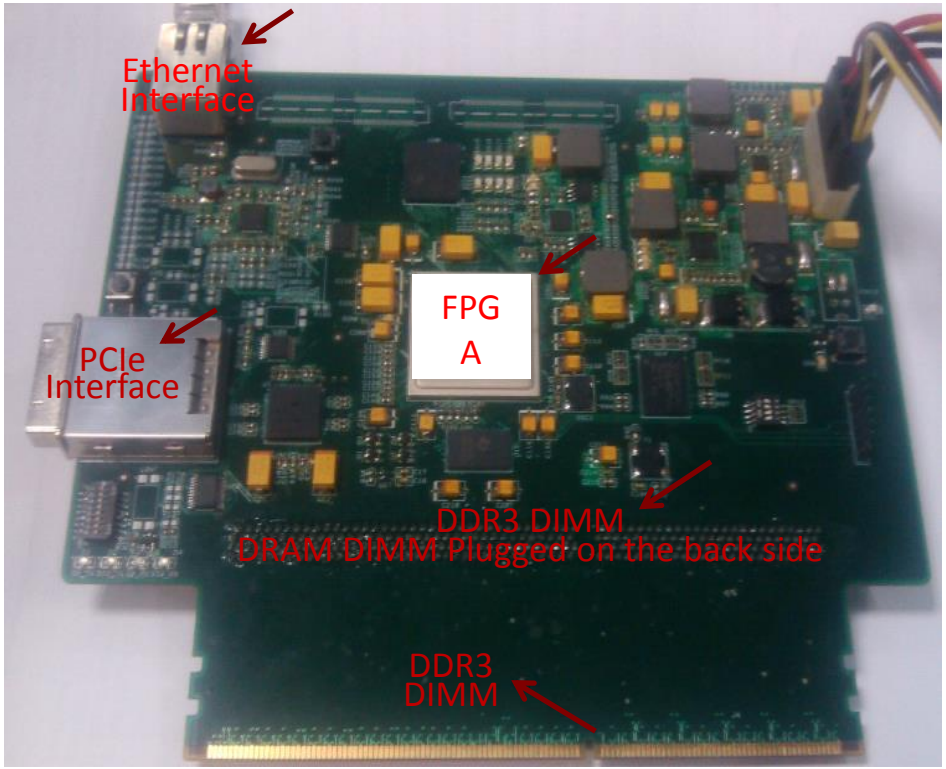
- HMTT: Hybrid Memory Trace Toolkit
  - A DDR3 SDRAM compatible memory monitoring system
  - Adopts hardware snooping technology

Memory Trace:

- Fine granularity: cache block
- <time_stamp, r/w, phy_addr>

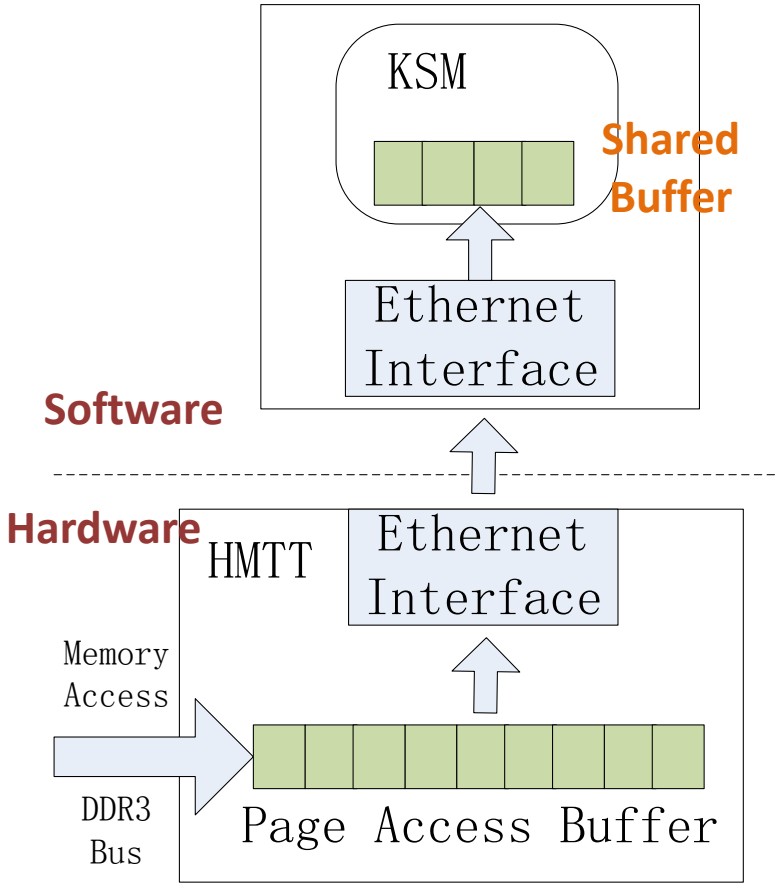
Advantages:

- Platform independent
- Negligible overhead
- Full-system real memory traces



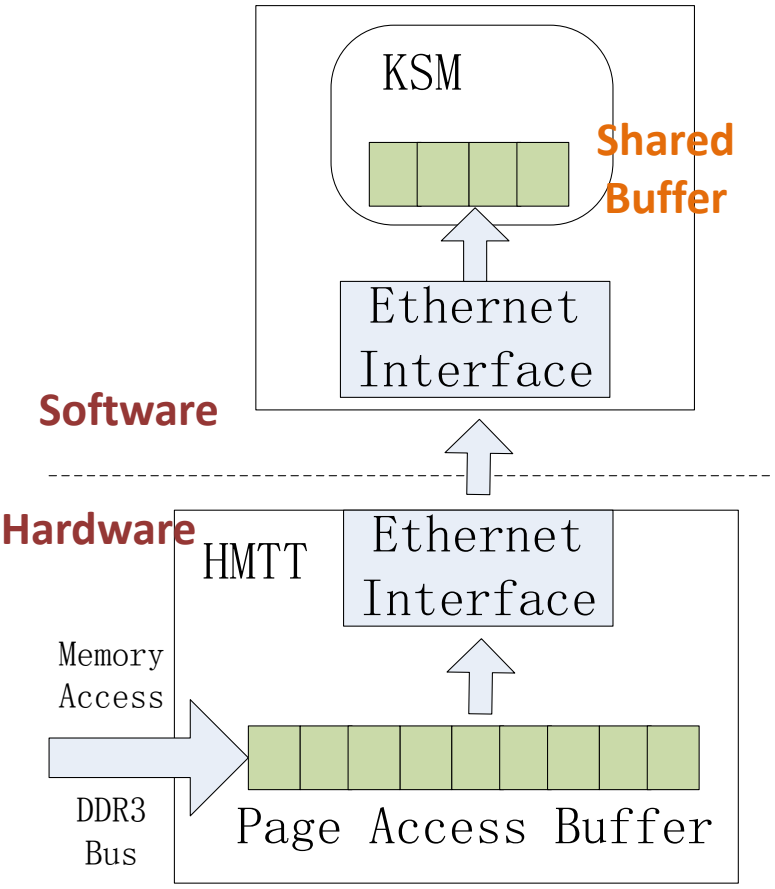
Pace Access Characteristics

- Page Access Characteristics are maintained by the HMTT
  - E.g. write access count of a page, write distribution of sub-pages
  - Implement a Page Access Buffer on the HMTT
  - Updated in the buffer when a memory access is monitored



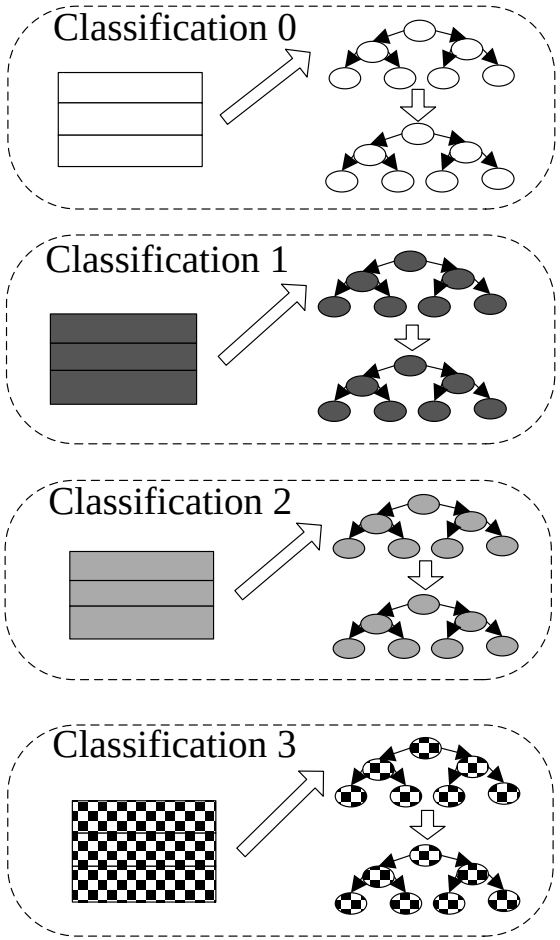
Pace Access Characteristics

- Page Access Characteristics are maintained by the HMTT
  - E.g. write access count of a page, write distribution of sub-pages
  - Implement a Page Access Buffer on the HMTT
  - Updated in the buffer when a memory access is monitored
- Fed back to the software (KSM) periodically through Ethernet Interface
  - We have implemented a shared (software) buffer as a kernel module
  - The KSM thread can utilize it to perform page classification and CMD



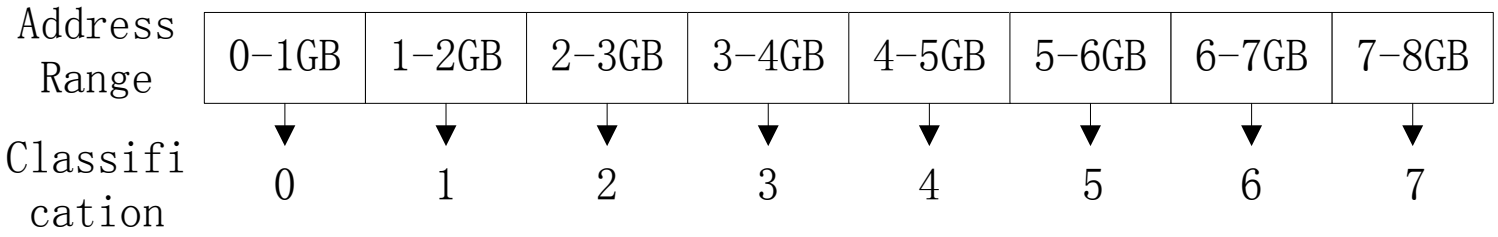
Classification on KSM

- The large global comparison trees are divided into multiple small trees
  - Pages are grouped into classifications based on page access characteristics
  - Local comparison trees dedicated to each page classification
  - Pages are just compared with nodes in the local trees
  - Pages from different classifications are never compared, probably result in futile comparisons



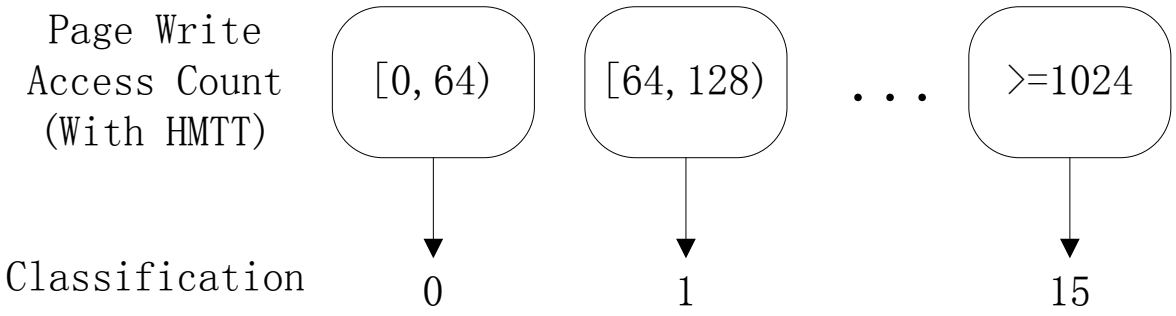
Page Classification Approaches

- We have implemented 3 different page classification approaches
- 1. CMD_Address (CMD_Addr):
  - Pages are classified based on physical address (pfn)
  - Static and simple, but page-access unaware
  - E.g. 8GB memory is evenly divided into 8 classifications



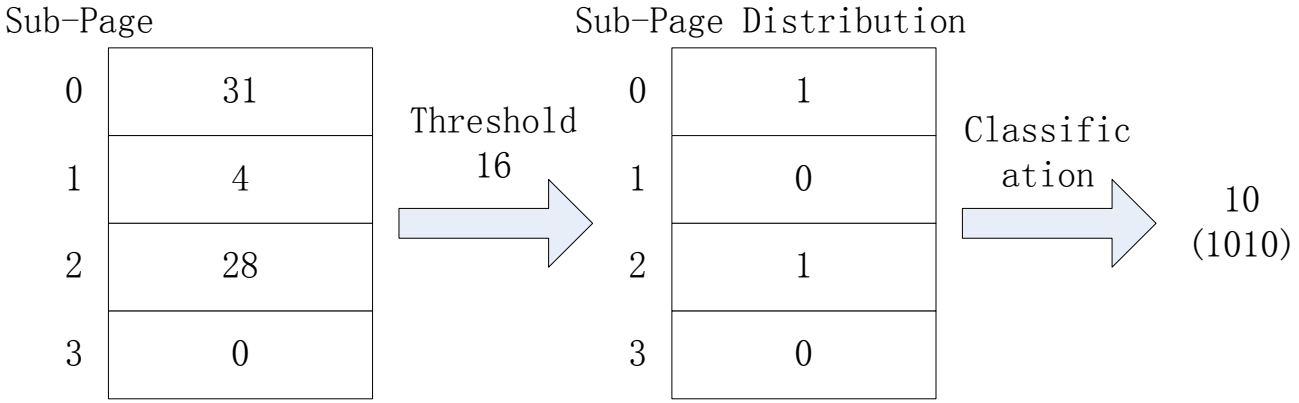
Page Classification Approaches

- 2. CMD_PageCount:
  - Pages are classified based on write access count.
  - Write access modifies page content and thus affects page sharing opportunities.
  - Page-access aware, slightly improve page classification accuracy, but still coarse granularity
  - E.g. page count threshold is set to 64

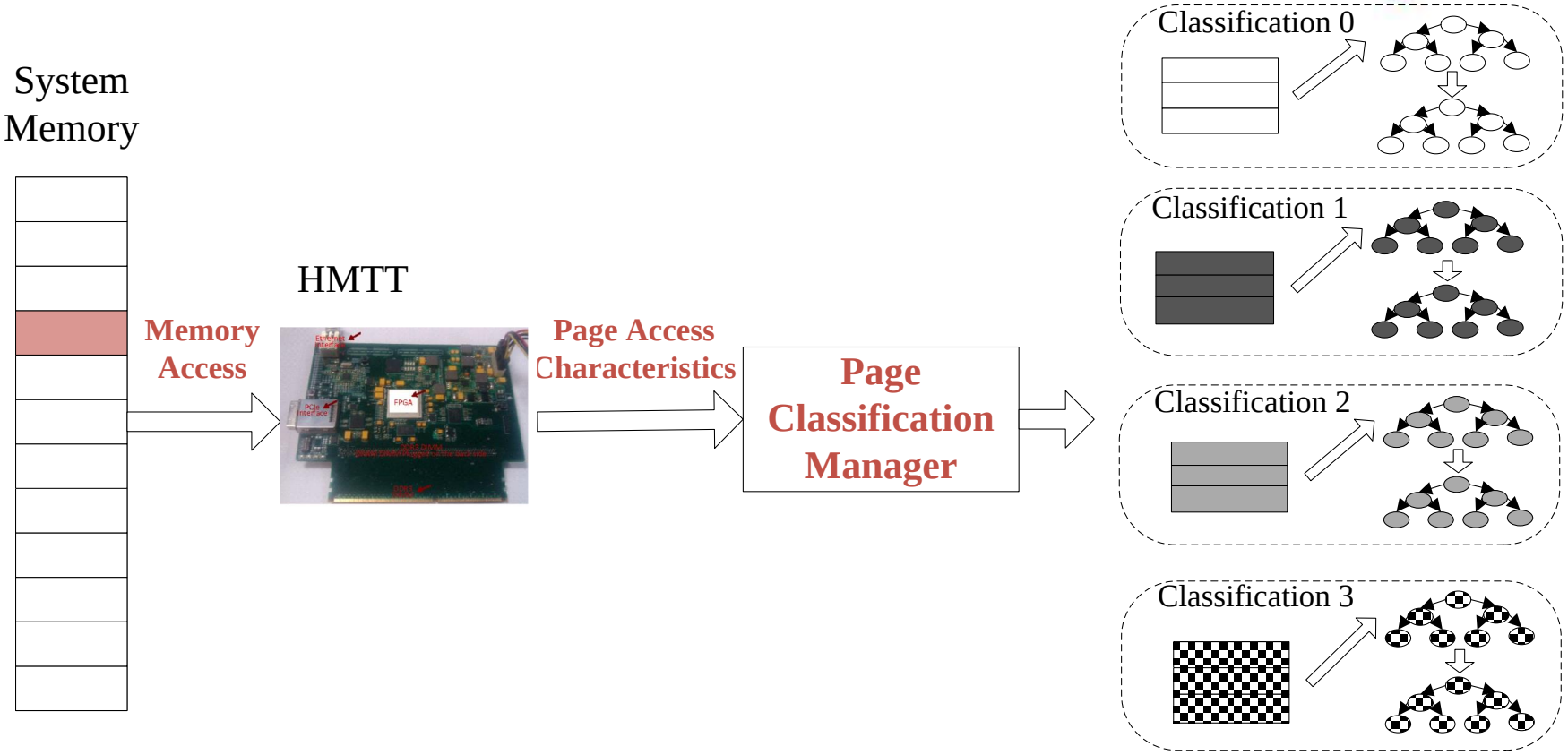


Page Classification Approaches

- 3. CMD_Subpage_Distribution:
  - Pages are divided into multiple sub-pages, and we monitor write access count on sub-page granularity
  - Pages are classified based on the write distribution of sub-pages
  - Fine granularity, improve classification accuracy
  - E.g. 4 sub-pages with write threshold of 16



Put them all together



Outline

- Background & Motivation
  - The profiling of KSM
- CMD: **C**lassification based **M**emory **D**eduplication
- Experimental Results
- Conclusion

Experimental Methodology

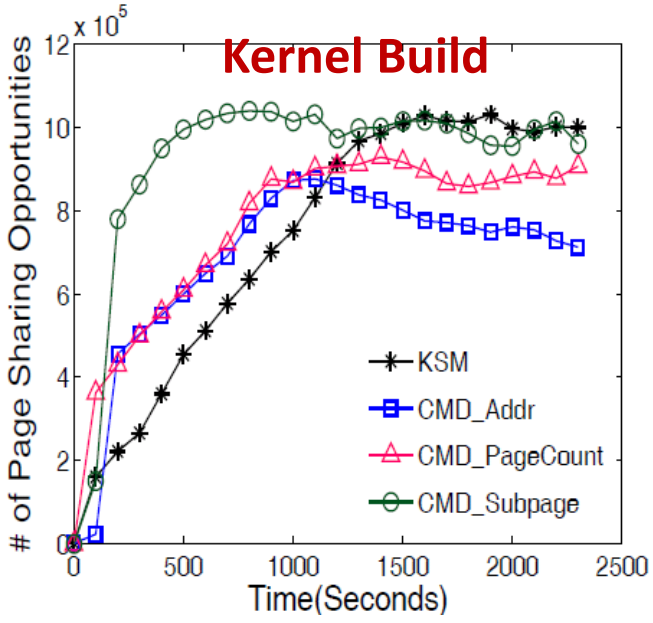
- Intel Xeon E5504 processor (2GHz)
  - 4 physical cores with Hyper-Thread disabled
  - 3-level cache, 16-way 4MB shared L3 cache
- Dual-ranked DDR3-800MHz physical memory, 8GB capacity in total

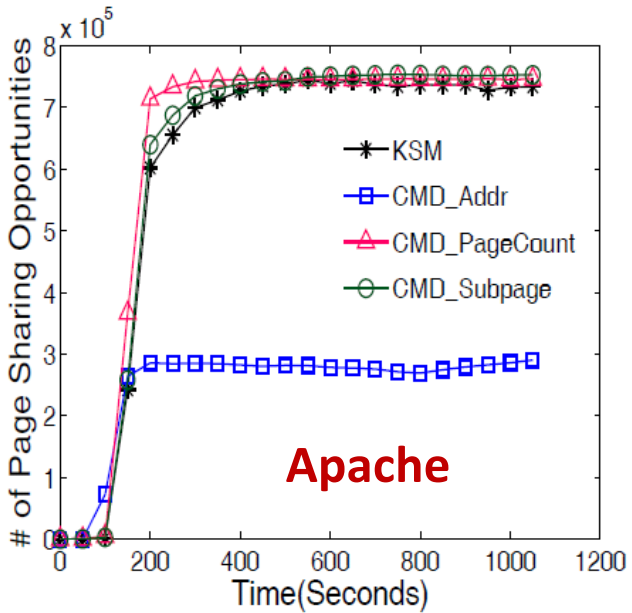
Experimental Methodology

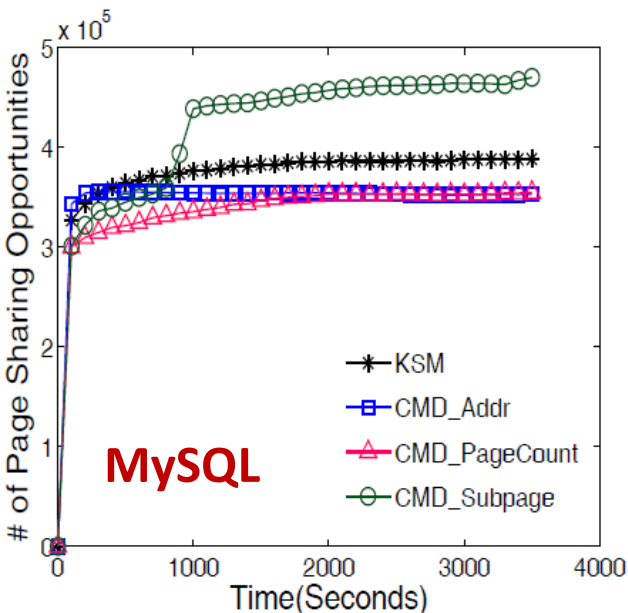
- Intel Xeon E5504 processor (2GHz)
  - 4 physical cores with Hyper-Thread disabled
  - 3-level cache, 16-way 4MB shared L3 cache
- Dual-ranked DDR3-800MHz physical memory, 8GB capacity in total
- Host server: CentOS 6.2 with Linux kernel 3.6.10 (implement CMD)
- We adopt libpcap to capture fed-back Ethernet packets from HMTT
- QEMU with KVM (qemu-kvm-1.2.0) to support guest VMs:
  - 4 VMs as default, each with 1 virtual CPU and 2GB memory
- Guest VMs: CentOS 6.3 with Linux kernel 2.6.32-279
- Workloads: Kernel Build, Apache (ab), MySQL (SysBench)

Page Sharing Opportunities

- CMD_Addr fails to detect page sharing opportunities, it is worst for Apache with ~39% compared with KSM
- CMD_PageCount is medium, it is about 87% of KSM for Kernel Build workload
- CMD_Subpage has the best ability to detect page sharing opportunities, it can even detect more page sharing for the MySQL workload
  - Because it can detect more short-lived page sharing

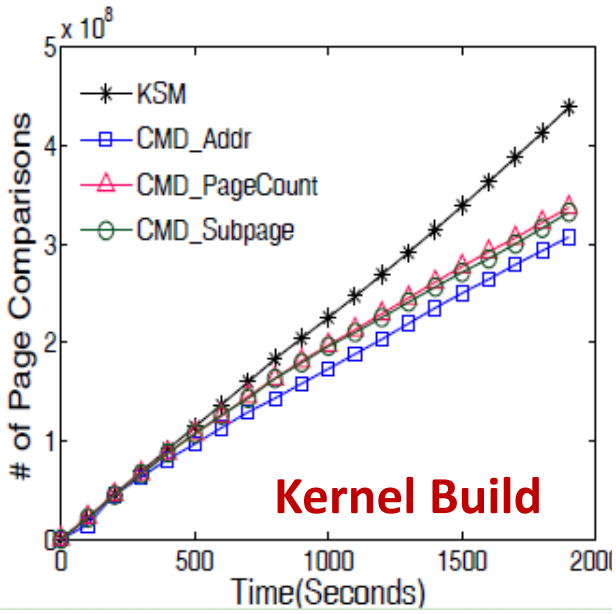


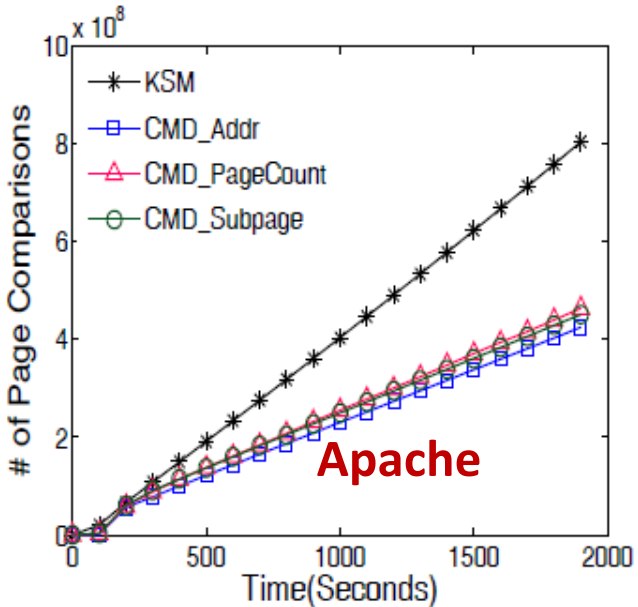


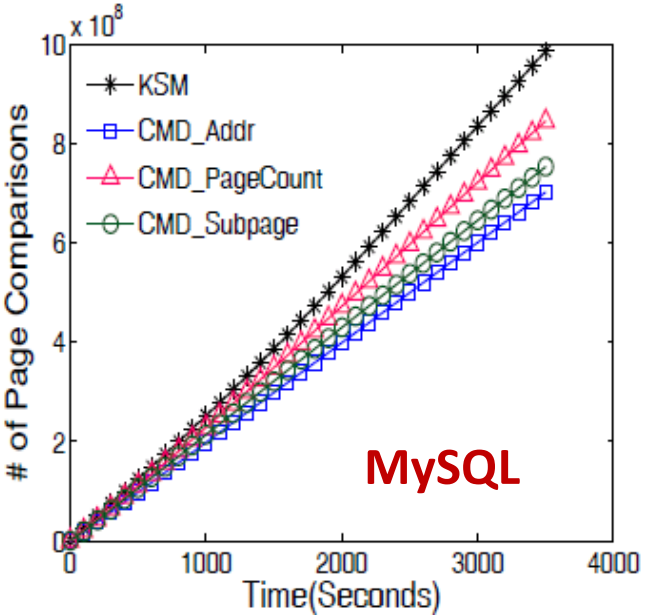


Page Comparisons

- CMD_Addr can reduce the most page comparisons, because it divides the global trees in most balance
- CMD_Subpage can also effectively reduce page comparisons

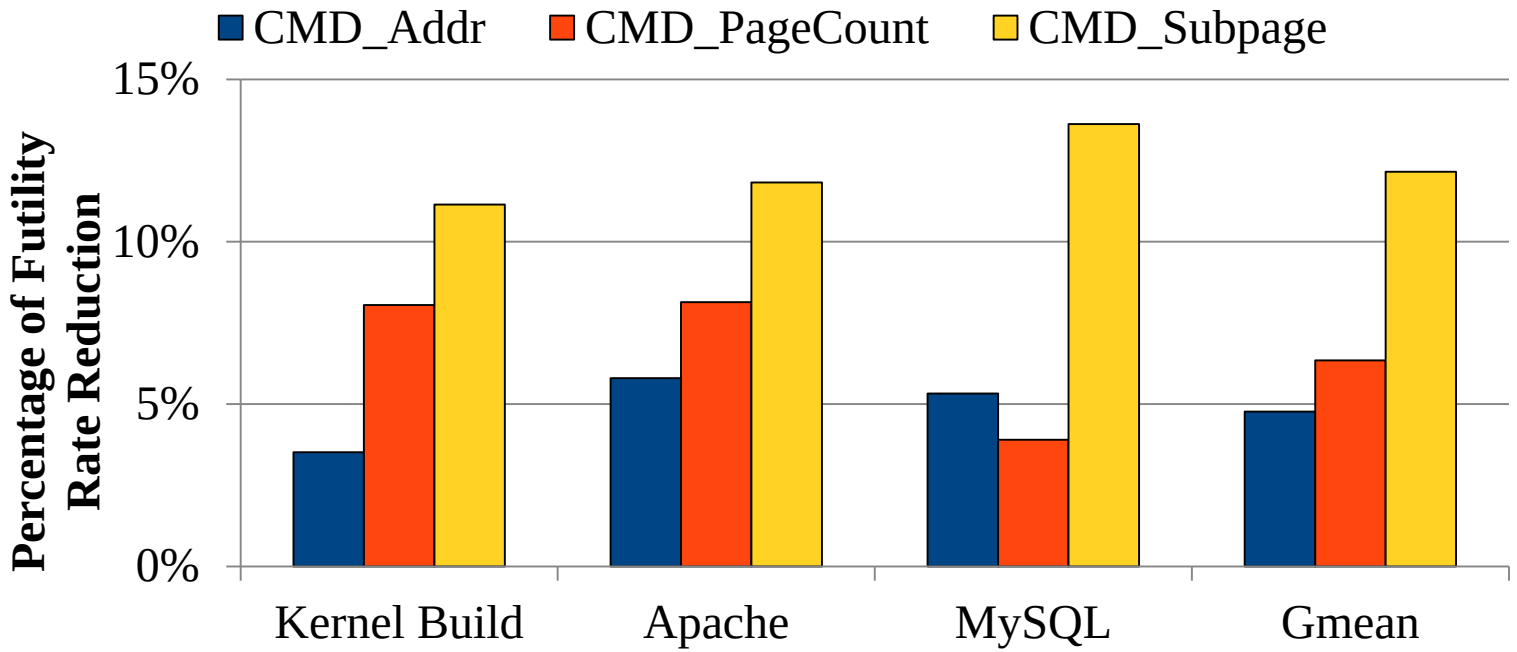






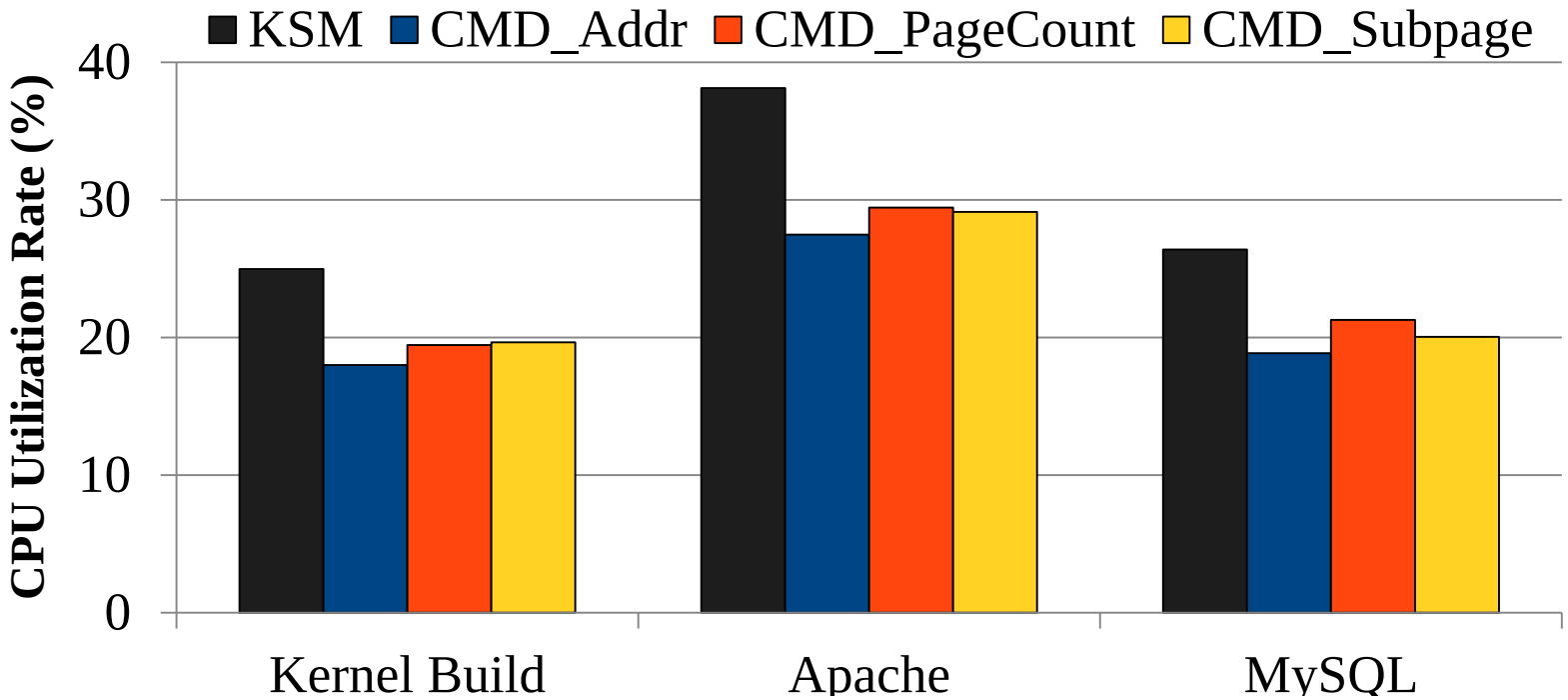
Futile Rate Reduction

- CMD_Addr can reduce the least futile rate by about 4.8%
- CMD_PageCount can reduce by about 6.4%
- CMD_Subpage can reduce the most by about 12%
  - But it still has space to find the best page classification approach



CPU Utilization Reduction

- The CPU Utilization of the KSM thread (ksmd) is got from top measurements taken every second
- All of the three approaches can reduce CPU Utilization because of the reduction of page comparisons



Outline

- Background & Motivation
  - The profiling of KSM
- CMD: **C**lassification based **M**emory **D**eduplication
- Experimental Results
- Conclusion

Conclusion

- We perform a detailed profiling of KSM
  - Page comparison contributes a certain portion of the overall KSM run-time overhead
  - There exists massive futile comparisons because of adopting two large global comparison trees

Conclusion

- We perform a detailed profiling of KSM
  - Page comparison contributes a certain portion of the overall KSM run-time overhead
  - There exists massive futile comparisons because of adopting two large global comparison trees
- We propose a lightweight approach called CMD
  - Pages are divided into different classifications based on page access characteristics with the help of HMTT
  - It maintains local comparison trees dedicated to each page classification, and pages comparisons are just performed in local.
  - CMD can reduce futile comparisons, meanwhile detect page sharing opportunities effectively

HMTT Homepage: http://asg.ict.ac.cn/hmtt/

Thanks !

&*Questions?*