Memory Sharing Based Virtual Machine Consolidation

Deependra Patel 120050032
Jayesh Bageriya 120050022

Introduction

Content-based page sharing is commonly used by many data center virtualization solutions like KVM, VMware ESX etc, to save memory. Identical pages of different virtual machines on same physical host can be consolidated to single shared page. In a multi-server data center, opportunities for sharing may be lost because the VMs holding identical pages are resident on different hosts. In order to obtain the full benefit of content-based page sharing it is necessary to place virtual machines such that VMs with similar memory content are located on the same hosts.

This system includes a memory fingerprinting system to efficiently determine the sharing potential among a set of VMs, and compute more efficient placements. We have implemented a prototype Memory Buddies system with VMware ESX Server and present experimental results on our testbed, as well as an analysis of an extensive memory trace study.

Methodology

1. Memory Tracer : A hash daemon keeps running on guest VM which regularly generates then sends hash list to the control server. Hsieh's Super Fast hash algorithm is used which generates 32 bit hash per page.
2. Control Server : Listens to hashes being sent to it. Stores these hash files. Analyses the VMs to decide colocation.
3. Determining the best possible colocation of VMs on physical machines - We have used a greedy heuristic approach for this problem as the problem of just determining a possible colocation or even determining whether if there exists a possible colocation is a NP-hard problem.
   The algorithm - We start with the physical machine which has the highest amount of memory. Then two VMs which have the highest sharing potential are chosen and placed in the currently selected physical machine. Next, we choose a VM which will produce highest expected sharing. The expected sharing of a VM is calculated with the help of following heuristic for kth VM on machine m -

   $$\text{expected\_sharing(k,m)} \quad \sum_{i=1}^{n} share(i,k)/i \quad i \in machine(m)$$

   An example of this - Consider a physical machine with two VMs in it and we are calculating the sharing potential of a third VM on this machine. Consider the sharing between 1st and 3rd to be 40 pages and that between 2nd and 3rd to be 30 pages. Hence the sharing potential on this machine would be between 40 and 70 (40 + 30) as

there would be minimum of 40 pages shared and maximum of 70. Our expected sharing heuristic would give the result of this to be 55.
If the VM chosen to be colocated on current physical machine needs more memory than available, then, the next largest physical machine is selected and all the above steps are repeated till all VMs are colocated.
We have made some reasonably practical assumption such as a physical machine is large enough to accommodate two VMs

Implementation

Memory Tracer

The python daemon (hashDaemon.py) performs following steps:

1. Inserts a kernel module (hash.ko) In the kernel module's init function, we go through all the pages of the memory of the VM to calculate 32 bit hash of each page and then write to the hash file. We only compute hash of the pages whose **page_count**>0 or the page is being used by some process. After all the pages are finished, file is closed.
2. Removes the module.
3. Makes connection to the control server.
4. Opens binary hash file (hash.bin), converts into integers to be sent over the network.
5. Close connection
6. Sleeps for some interval, after that repeat from 1

Control Server

It has 3 components.

a. Server to receive hash files (server.py). This keeps listening for the incoming hash files. Stores them in format VM[VM Number].txt[version number]. eg VM1.txt0, VM1.txt1, VM2.txt2 etc
b. Analyser(analyse.py) to compare hashes of any two files to output number of pages matched. This first sorts both the files then compares.
c. Determining best colocation (best_matching.py). All physical machine are kept sorted in descending order on basis of memory and chosen one by one. Analyser is used to get the sharing potential between all two pairs of VMs. Based on this, expected sharing is calculated as per the described algorithm and the one with largest sharing potential is chosen to be colocated on the machine

Experimentation/Results

1.Effect of hashing on performance

We ran CPU Blowfish, a standard cpu performance benchmark to see effect on performance. The test was run on Ubuntu 14.04 with 2GB memory.

	CPU Blowfish Score
Without hashing	8.50
With hashing	10.36

There is around 18% drop in the performance which is acceptable.

2.Time taken for hashing

It takes ~2 seconds for generating the hash of entire used memory in a VM. The file generated is ~4Mb in size, so ~3 seconds is taken in sending the file to the control server. Most of the time is spent in writing to the hash file.

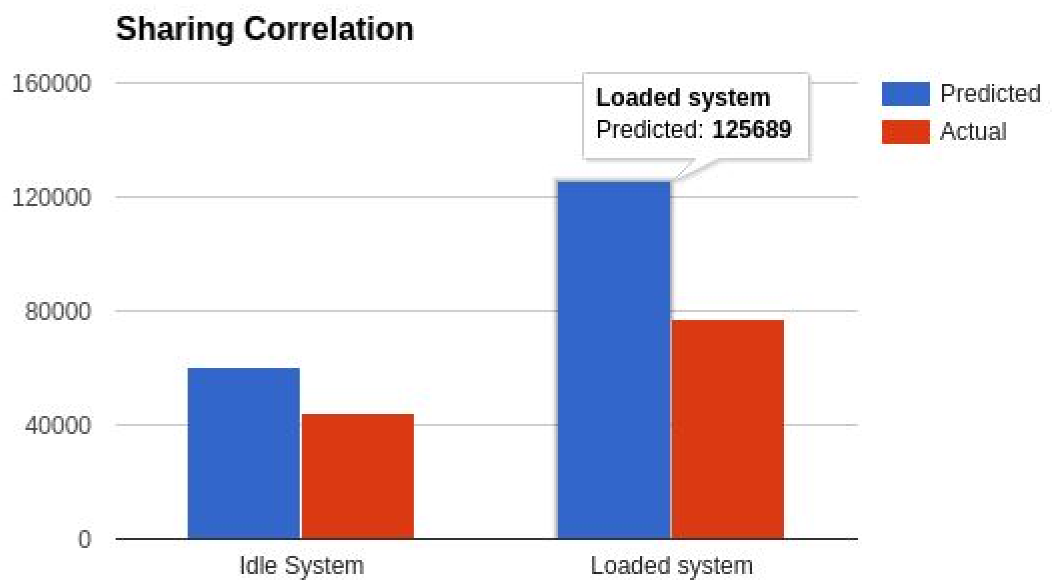
3.Correctness of the fingerprint generated

Setup :
Idle System case - 2 VMs running Ubuntu 14.04 each with 2 GB memory with idle system, 910 MB and 1092 MB being actually used.
Loaded System case - 2 VMs running Ubuntu 14.04 each with 2 GB memroy with firefox(also with youtube video), libre office and nautilius open. 1758 MB and 1807 MB being actually used.

KVM has KSM module which shares pages among VMs. Change default pages_to_scan value of 100 to 20000, also change sleep_millisecs from 200 to 50. This was done for quick sharing of pages if possible.

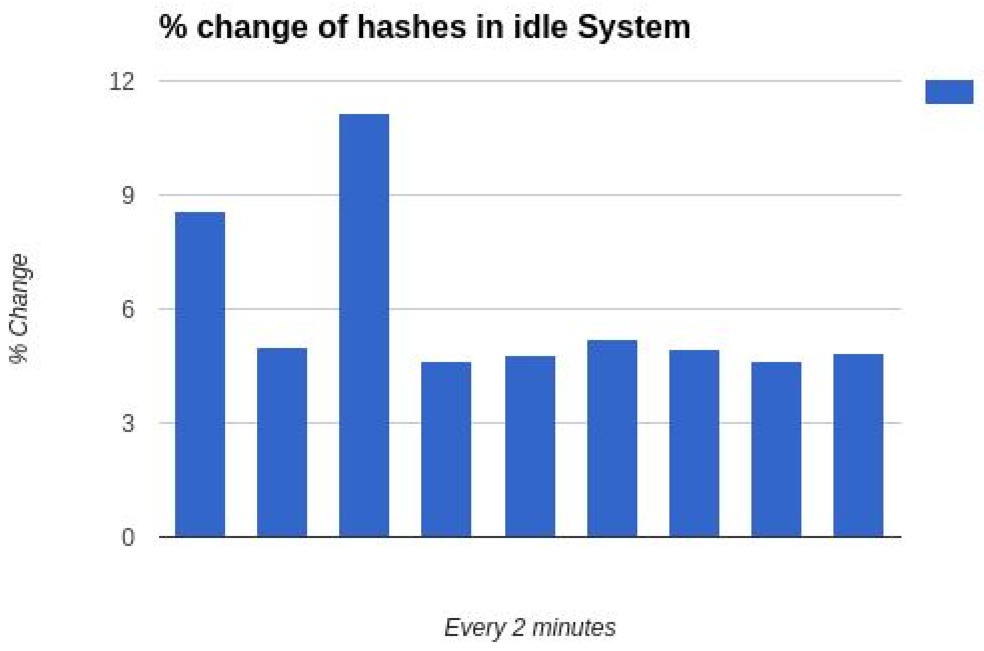
To display the actual number of pages shared,

```
sudo cat /sys/kernel/mm/ksm/pages_shared
```

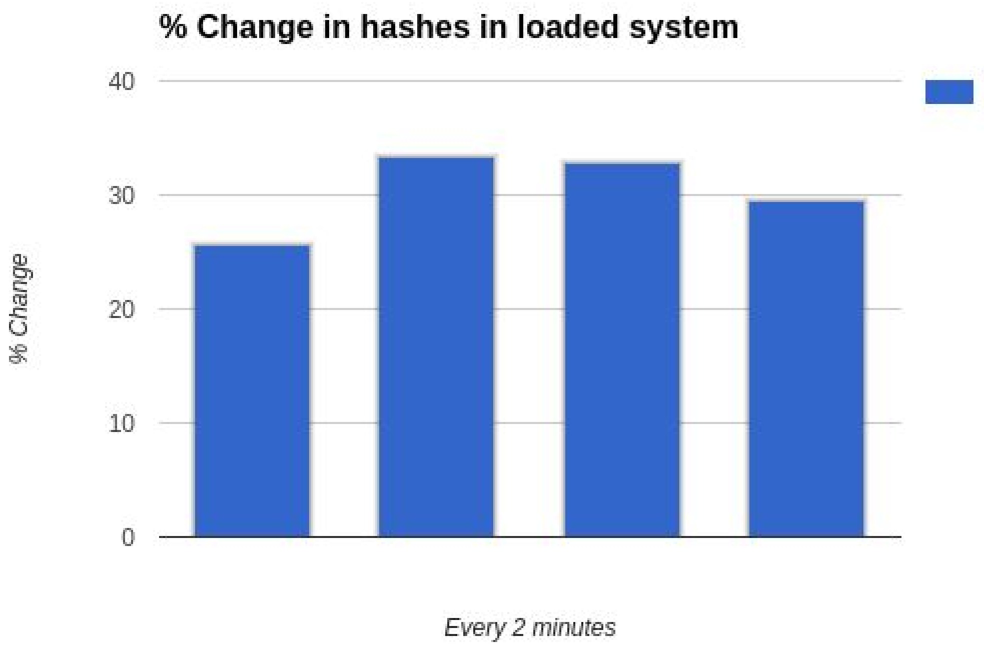


4. Change of fingerprints with time

Idle System case - VM running Ubuntu 14.04 each with 2 GB memory with idle system.



Loaded System case - VM running Ubuntu 14.04 each with 2 GB memory with firefox(also with youtube video), libre office and nautilius open.



4. Co-location Algorithm correctness (best_matching.py)

Input:

	Physical Machine
VM0 (1GB)	M0 (3GB)
VM1 (2GB)	M0
VM2 (1GB)	M1 (3GB)
VM3 (2GB)	M1
VM4 (1GB)	M2 (3GB)

Output :

	Physical Machine
VM0 (1GB)	M1 (3GB)
VM1 (2GB)	M1
VM2 (1GB)	M0 (3GB)
VM3 (2GB)	M2
VM4 (1GB)	M0 (3GB)

Results -

Pairwise common distinct hashes among various VMs,

	VM0	VM1	VM2	VM3	VM4
VM0		254 MB	235 MB	33 MB	207 MB
VM1			121 MB	33 MB	114 MB
VM2				12 MB	339 MB
VM3					10 MB
VM4					

```
Memory usage before colocation on physical machine # 0  is  1536 MB
Memory usage before colocation on physical machine # 1  is  2357 MB
Memory usage before colocation on physical machine # 2  is  857 MB
Initial usage was 4750 MB

Memory usage after colocation on physical machine # 0  is  1331 MB
Memory usage after colocation on physical machine # 1  is  1536 MB
Memory usage after colocation on physical machine # 2  is  1556 MB
After colocation memory usage was 4423 MB
Memory saved is  327 MB
```

Future Work / Improvements

1. Use Bloom filter instead of super fast hash to reduce network usage and cpu utilization
2. Add functionality to actually migrate VMs using our co-location heuristic
3. Calculate hashes in KVM will not require modifying the OS
4. Better heuristic can be added for VM colocation