On Exploiting Page Sharing in a Virtualized Environment - an Empirical Study of Virtualization Versus Lightweight Containers

Ashish Sonone, Anand Soni, Senthil Nathan, Umesh Bellur
Department of Computer Science and Engineering
IIT Bombay
Mumbai, India
ashishsonone, anandsoni, cendhu, umesh@cse.iitb.ac.in

***Abstract*—While virtualized solutions are firmly entrenched in cloud data centers to provide isolated execution environments, the chief overhead it suffers from is that of memory consumption. Even pages that are common in multiple virtual machines (VMs) on the same physical machine (PM) are not shared and multiple copies exist thereby draining valuable memory resources and capping the number of VMs that can be instantiated on a PM. Lightweight containers (LWCs) on the other hand does not suffer from this situation since virtually everything is shared via Copy on Write semantics. As a result the capacity of a PM to host LWCs is far higher than hosting equivalent VMs. In this paper, we evaluate a solution using uKSM to exploit memory page redundancy amongst multiple VMs on a PM executing the same operating system thereby enabling a comparison of the two technologies as far as its capacity to instantiate multiple instances is concerned. We performed a thorough empirical evaluation of the two solutions and present comprehensive results.**

***Keywords*-cloud, virtual machine, light weight container, KSM**

I. Introduction

Data centers host multiple applications on a single physical machine to increase resource utilization while providing performance isolation with the help of either virtual machines (VM) or containers. Resources such as CPU, memory, disk and network bandwidth can be allocated to each VM or container according to the need of application hosted on them.Containers are light weight in terms of memory utilization and code execution path compared to virtual machines as the application hosted inside a container executes directly on the host operating system (OS). In the case of a virtual machine, the application is hosted on the guest OS which runs on the virtual machine. The hypervisor & host OS aids in sharing resources between virtual machines. The widely used hypervisor that enables the execution of virtual machines are Xen [1], KVM [2], & VMware [3] and widely used containers are LXC [4], openVZ [5] and VServer [6].

VMs and containers can be compared on the basis of their (i) functionalities, (ii) performance and (iii) resource utilization overhead (mainly memory footprint).

Functionality. As applications hosted on containers share the host OS, crashing of the host OS's kernel by any faulty application can result in the failure of all applications executing on that machine. On the other hand, each virtual machine runs its own OS and hence, a faulty application cannot crash the host OS. Further, each application can run on any customized and optimized guest OS which is different from the host OS. For example, each guest OS can run its own I/O scheduler which is optimized for the hosted application. In addition to this, the hypervisor provides many other features such as live migration of virtual machines, high availability using periodic checkpointing and storage migration.

Performance. For virtual machines, the hardware resources need to be virtualized so that guest OS can run on a virtual machine. The virtualization of memory, disk and network resources are handled by the hypervisor which adds some overheads in the execution of the application. Thus, the performance of an application executing on virtual machine degrades compared to the bare-metal machine. On the other hand, a container executes directly on the host OS without any virtualization overhead. Prior research [7], [8], [9], [10], [11], [12] has compared the performance of both virtual machine and container using micro and macro benchmarks. The performance of an application executing on containers was close to it's performance on the host OS whereas there was a noticeable performance degradation when executing on a VM. However, recent advancements in providing hardware support for virtualization has resulted in significantly denting the performance overheads of virtualized applications. For example, Intel provides certain hardware features such as EPT [13] for memory virtualization and SRIOV [14] for network virtualization.

Resource utilization overhead. Memory is a critical resource in a data center which limits the number of applications that can be hosted on a physical machine and hence limits the profit. hosted application and hence the profit. The amount of memory utilized by the virtual machine for a given application would be higher than the container due to the additional memory occupied by guest OS. As a result, if applications are hosted on containers rather than virtual machines, a data center can host a few additional applications. However, this would take away the functionalities provided

by the virtual machine. To avoid this, we should try to reduce the memory utilization of VMs. With each VM running the same guest OS, the chance of finding identical content in memory pages are quite high. Hence, one way to reduce the memory utilization is to replace redundant memory pages with a single write protected page.

KSM [15] and uKSM [16] are tools which periodically scans userspace memory to find memory pages with identical content and keeps only one copy of redundant pages to save memory. In other words, these tools find the duplicate memory pages between processes and replace these pages with a single write protected page (which is also knows as Copy on Write). In case of KVM and LXC, each virtual machine and container executes as a user process and hence the memory sharing technique can be employed to reduce the memory footprint. None of the existing literature have compared the memory utilization of both KVM's VM and LXC containers with and without a memory sharing technique. Our goal is to perform a comprehensive study to understand the effectivenss of the memory sharing technique on the memory footprint of both KVM's virtual machine and LXC container. To be specific, our goal is to answer the following questions

1) Can memory sharing technique reduce the memory footprint of VMs such that the memory utilization of VMs and containers is comparable?
2) Does the execution of a workload on the application in either of these technologies reduce the memory sharing opportunity?
3) What is the impact of memory sharing technique on the performance of application?
4) Is the time taken to boot n VMs significantly higher than the booting time of n containers?
5) What is the cost paid in terms of resource utilization during boot time?

The relevance of the last two questions has to do with the fact that no such comparison exists and is an important factor in dealing with large boot storms in data center. We believe that it's necessary to round out the comparison of containers and VMs as competing technologies for co-hosting applications on PMs in a data center.

The rest of the paper is structured as follows: Section II lists existing work which compared the performance of application executing on containers and virtual machines. Section III describes our experimental methodology and setup used to answer our questions. Section IV presents the results of our comprehensive study while Section V concludes this paper.

II. Related Work

Table 1 captures previous performance studies that have sought to compare containers and virtual machines. As can be seen, none of the studies looked to compare memory utilization of VMs with containers. In addition, there has not been any reported work on using KSM like memory sharing techniques to reduce the memory footprint of a set of VMs executing on a PM.

The work in [7] compared the performance of applications running on LXC container and KVM/Xen's virtual machine. The performance of CPU intensive applications and network intensive applications were observed to be same in both containers and VMs. However, a significant degradation is observed for both memory intensive and disk intensive applications with VMs due to the virtualization overhead of the respective resources.

[8] studied the performance of WordPress blog executing on both Xen's VM and LXC container. The throughput of WordPress blog with containers was observed to be 8000 requests per second higher than the VM. This work also qualitatively compare the operational flexibility in terms of image creation time, startup time, high availability and migration.

The study in [9] compared the performance of micro benchmarks executing on containers such as openVZ, docker and lmctfy with KVM's virtual machine. The performance of CPU intensive application was the same on all platforms but memory and disk intensive application observed a significant reduction in the performance while executing on a virtual machine. Further, the performance of network intensive application also degraded when using large packet sizes. However, this impact was not observed in [7] as they have not varied the packet size. The work in [10] compared the performance of HPC application executing on containers such as openVZ, VServer, and LXC with Xen's VM. The observation listed in this work is similar to [9]. Further, the observations made in [11] and [12] are also similar to [9], [10].

All existing work only compared VMs and containers on the basis of their performance but not in terms of memory utilization with and without uKSM (i.e., memory sharing tool). uKSM scans memory pages and compares its content to find pages with identical content. On detecting identical pages, a single copy of the page is shared and the memory occupied by other pages are freed, thereby reducing the memory utilization. The shared page is write protected. A write to the shared page by a process is handled by providing an exclusive copy to of the page to that process. Though uKSM reduces the memory utilization, additional CPU resource is consumed for scanning and comparing memory pages. In our work, we mainly compare the memory utilization of KVM's virtual machine and LXC container in the presence of uKSM functionality. Though several literature [17], [18], [19] have quantified the memory sharing opportunity between VMs using KSM, they do not answer fives questions listed in Section I.

Table I
PERFORMANCE AND COST COMPARISON OF VIRTUAL MACHINES AND CONTAINERS

Paper	Virtual Machines		Containers			Benchmark					Memory Usage (with-/without uKSM)
	KVM	*Xen*	*LXC*	*openVZ*	*VServer*	*CPU*	*Memory*	*Network*	*Disk*	*Macro*	
[7]	✓	✓	✓			✓	✓	✓	✓	✓	×
[8]		✓	✓							✓	×
[9]	✓			✓		✓	✓	✓	✓		×
[10]		✓	✓	✓	✓	✓	✓	✓	✓		×
[11]	✓	✓		✓		✓	✓	✓	✓	✓	×
[12]	✓		✓			✓	✓	✓	✓	✓	×
our work	✓		✓							✓	✓

III. METHODOLOGY AND EXPERIMENTAL SETUP

A. Experimental Setup

All experiments were performed on a blade server with 8 CPU cores & 24 GB RAM executing Ubuntu 12.04. The linux kernel (v3.8.13.25) was patched with uKSM version 0.1.2.2. Experiments were performed on KVM and LXC containers. KVMs were configured with 256 MB RAM and 1 VCPU each.

B. Methodology

Experiments were broadly classified into idle and non-idle scenarios :

- Idle : Multiple KVMs/LXCs were run with mysql or apache or a mix of the two (some running apache, some mysql) but without any workload. Our goal was to find the maximum amount of sharing possible with uKSM for both LXC and KVM such that the maximum sharing can serve as a reference when workload is executed later.
- Non-idle : OLTP benchmarks were run to generate mysql database workload. We intended to determine whether the amount of sharing increases or decreases due to load. Here is a brief description of different oltp benchmarks that were used :
  - Epinions : This workload is centered around users interacting with other users and writing reviews for various items in the database (e.g., products). It consists of nine different transactions, of which four interact only with user records, four interact only with item records, and one that interacts with all of the tables in the database.
  - SEATS : The SEATS benchmark models an airline ticketing system where customers search for flights and make online reservations. It consists of eight tables and six transaction types.
  - TPCC : It consists of nine tables and five procedures that simulate a warehouse-centric order processing application. TPCC's transactions are more complex and write-heavy than in other benchmarks.
  - YCSB : The YCSB workload contains various combinations of read/write operations and access distributions that match products inside Yahoo!. It is representative of very simple primary-key based key-value store applications.

General Procedure for the experiments : A set of 'n' KVM/LXC(s) are started with each running a single instance of either mysql or apache server. Once they are up and running, we collect data for a certain time period with uKSM turned off. Then, uKSM is turned on to collect data for a certain time period. We collect the following three metrics every second during every collection period.

1) Memory usage of host(using *free -m* command)
2) CPU usage of uksm (using *top* command)
3) Page sharing statistics for uskm (obtained from */sys/kernel/mm/uksm* folder)

IV. EMPIRICAL EVALUATION OF LXCS VERSUS VMS

In this paper, we have mentioned the memory utilization in terms of overhead over idle host's memory utilization. This is because, the memory utilization of idle host varies, and hence instead of host's total memory utilization, overhead gives a consistent numerical value to measure memory utilization across experiments that ran across several days.

First, we measured memory usage of a single LXC/KVM running apache or mysql server. The idle host (with no LXC/KVM running) takes around 560 MB memory. The following are the memory utilization overhead for different configurations:

- KVM with apache server : 234 MB
- LXC with apache server : 12.3 MB
- KVM with mysql server : 264 MB
- LXC with mysql server : 44 MB

A. Resource Cost with uKSM

To see how uKSM CPU utilization varies as uKSM performs memory page sharing, we plotted :

- Pages scanned by uKSM vs time

- uKSM CPU utilization vs time

Figure 1 shows the corelation between uKSM memory scanning and its CPU usage. As observed from the figure, we observed that the CPU usage is directly proportional to the rate at which pages are scanned by uKSM. Further, we observed that uKSM works in batches with periodic bursts of CPU utilization followed by scanning activity. uKSM does not affect application performance as separate CPU cores are assigned to VM and uKSM. Here are two terms we use

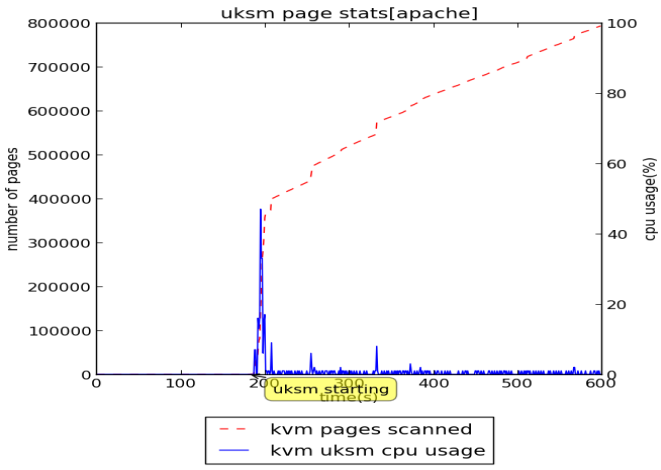

Figure 1. CPU usage of uKSM for five KVMs running Apache

frequently with regard to uKSM in this paper:

- uKSM pages shared : It is the number of pages that have been mapped by different page table entries, i.e. shared by different processes.
- uKSM memory pages sharing (*a.k.a.* saved): It is roughly the number of redundant pages you have saved, with uKSM.

In all the further experiments, five LXCs/KVMs were started together and monitored.

B. KVM/LXC running idle mysql server

As observed from Figure 2, when running mysql server on five LXCs, we recorded a memory overhead of 216 MB, approximately 5 times the overhead for one LXC running mysql server. On starting uKSM, the memory overhead dips down by around 100 MB and comes down to about 118 MB. On the other hand, five KVMs running mysql impose a memory overhead of around 1300 MB (as expected). uKSM brings the memory overhead down to 540 MB i.e a decrease of almost about half of the original overhead. A plausible explanantion is that multiple KVMs have a lot of kernel pages in common which were shared by uKSM.

Figure 3 shows that the number of uKSM memory pages sharing (or saved) increases from 1807 to 6718 and uKSM pages shared increases from 1135 to 1180 for LXC. We did not observe the 1:4 ratio of pages shared to memory pages sharing (or saved) because when uKSM starts, other system processes also contribute to sharing of pages which fades away the contribution due to LXC alone.

For KVM, uKSM pages shared increase from 1074 to around 31240 and uKSM memory pages sharing (or saved) from 1848 to around 131260. As expected, the ratio of uKSM pages shared to uKSM memory pages saved is 1:4 because for every five duplicate pages (across 5 KVMs), only one is kept after uKSM scans the memory and hence it reduces memory redundancy by 80 percent.

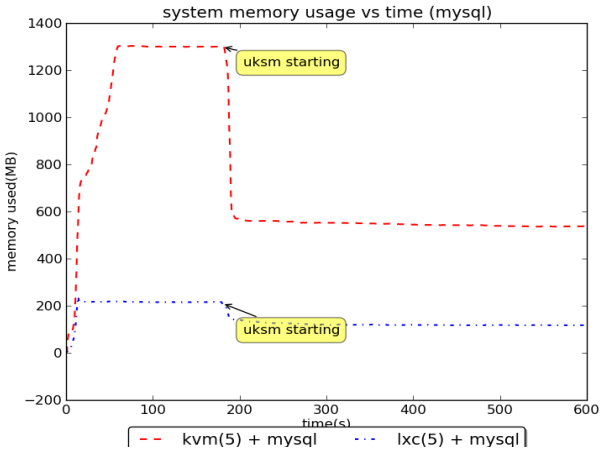

Figure 2. Memory overhead for five KVMs/LXCs running mysql

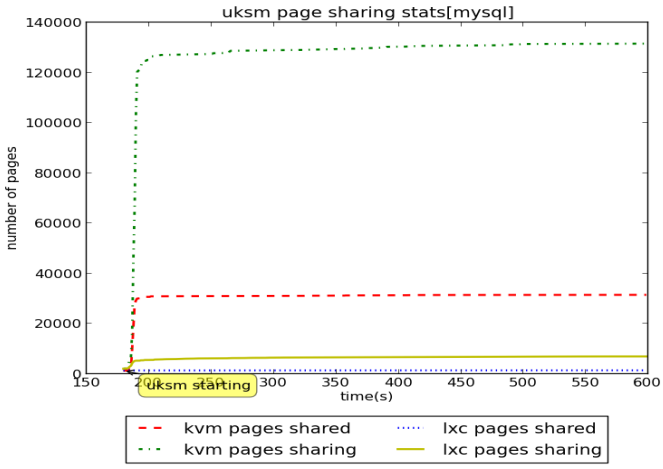

Figure 3. Page sharing statistics for five KVMs/LXCs running mysql

C. KVM/LXC(s) running idle apache server

For five LXCs running apache server, we expect around 60 MB memory overhead (around 12MB per LXC) with uKSM disabled. As shown in Figure 4, the experimental memory overhead of 59.33 MB met this expectation.

However, after uKSM was turned on, memory overhead increased unexpectedly to around 70 MB. A possible cause for this could be random jitter in the system.
With five KVMs running apache server, expected memory overhead was 1170 MB. Experimentally, we recorded a memory overhead of 1034 MB. This could be due to a common VMM which uses certain memory whether single or multiple KVM instances are running, and hence the overhead is not simply five-fold. After turning uKSM on, the overhead dips down to 441 MB (a decrease of 593 MB).

In case of KVM, as observable from Figure 5, the number of pages shared increased from 579 to 27428 and the number of memory pages sharing (or saved) from 1164 to 112446. In the case of LXC, pages shared increased from 1193 to 1334. Though pages shared increases, we observed an increase in memory usage which is just the opposite of what we expected. Similarly, memory pages sharing saved also increase from 1832 to 2231.

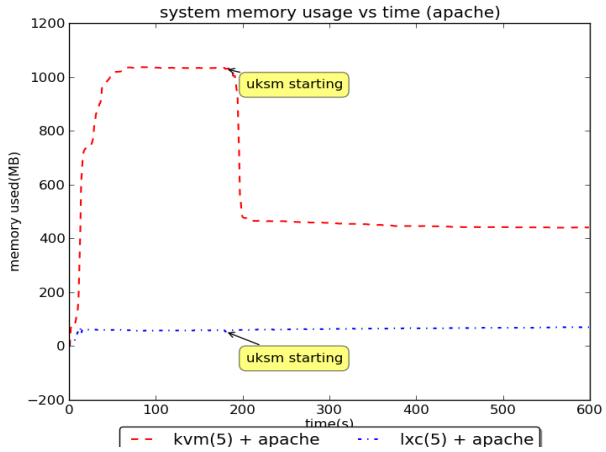

Figure 4. Memory overhead for five KVMs/LXCs running apache

D. KVM/LXC(s) running a mix of idle apache and mysql servers

In this experiment, we run three LXCs/KVMs each executing mysql server, and another two KVMs/LXCs each executing apache sever. We wanted to observe how uKSM performs when a mix of applications is running. From the three instances of KVM executing mysql server, we expected a contribution of 3×264 MB, and a contribution of 2×234 MB from the two instances executing apache server resulting in a total memory overhead of approximately 1260 MB. As observable from Figure 6, we recorded an actual overhead of around 1172 MB which is acceptable. Turning on uKSM brings down the overhead to 504 MB which is in between the recorded cut down in memory overhead when running five mysql servers and five apache servers. From the above observation, it can be stated that in the case of KVMs,

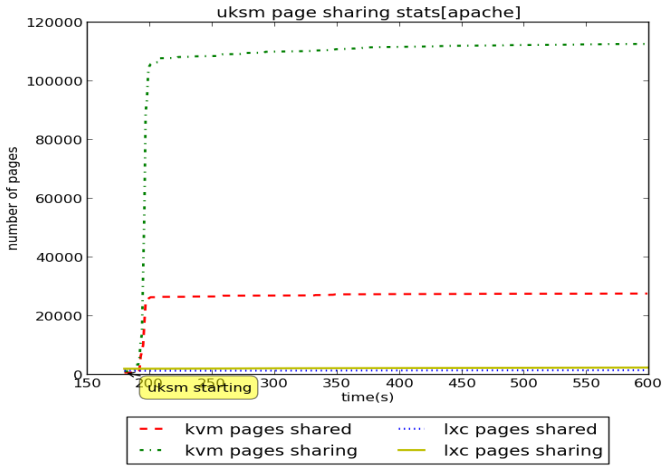

Figure 5. Page sharing statistics for five KVMs/LXCs running apache

most of the sharing happens due to kernel pages and the application has a small memory footprint. Hence, the amount of shareable memory is not severely affected by running a mix of different applications.

For LXC instances, on the other hand, a total memory overhead of 152 MB was recorded which was in accordance with the expected overhead value of 156.6 MB which includes a contribution of 3×44 MB (from the three LXC instances running mysql server) and 2×13.2 MB (from the two LXC instances running apache server). Turning on uKSM breaks down the memory overhead by 100 MB.

As depicted by Figure 7, for KVM, we observed an increase in pages shared from 1337 to 32309 and in memory pages sharing (or saved) from 2194 to 117449. Whereas, for LXCs, pages shared increases from 1411 to 1439 and memory pages sharing (or saved) from 2353 to 5170 which is similar to the case of running five mysql servers instead of a mix.

E. KVM/LXC(s) running mysql server with OLTP benchmark

In this experiment, we generate requests for the mysql servers executing in the LXC/KVM(s) using OLTP benchmarks. A separate blade server was used to run the OLTP benchmarks. Once all the LXC/KVM(s) have booted, we run the OLTP benchmark for five minutes with uKSM turned off and record certain metrics (as described in Section III-B) and collect OLTP throughput logs. We again run the benchmarks for another five minutes with uKSM turned on. We run five benchmarks; one targetted at each LXC/KVM.

As observable from Figure 8, with uKSM turned off, five LXC instances running mysql with OLTP benchmarks have a memory overhead of 918 MB. This is way higher than 216 MB, the memory footprint of five LXC instances executing idle mysql. This increase can be attributed to database being loaded in memory to serve the SQL queries (requests). Since

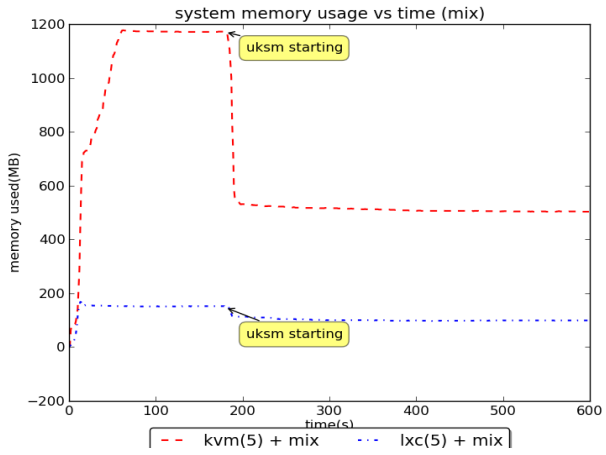

Figure 6. Memory overhead for five KVMs/LXCs running a mix of mysql and apache

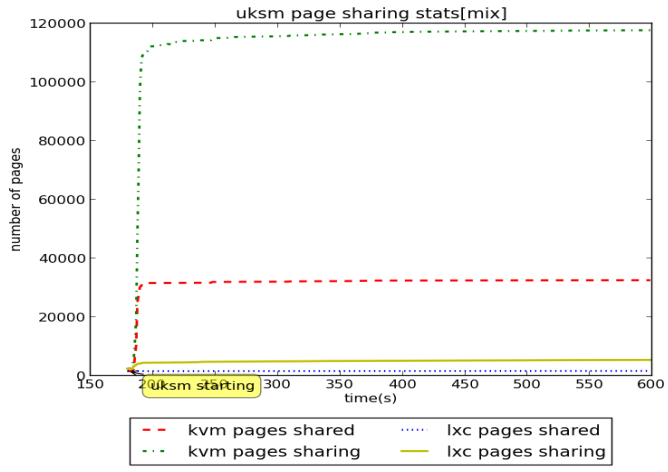

Figure 7. Page sharing statistics for five KVMs running a mix of mysql and apache

LXC is not allocated memory beforehand (LXC just has an upper cap on how much memory it can use), unlike KVM which uses all of its assigned memory (256 MB), it uses only the memory required to support the running processes. When database is loaded, more memory is allocated and used. Hence we see a significant increase of around 700 MB in the memory overhead. Turning on uKSM, brings down this overhead to 883 MB.

On the other hand, the five KVM instances (executing mysql) used around 1398 MB memory when OLTP benchmarks were run. This is around 100 MB higher than five KVMs running idle mysql. Turning uKSM on brought the overhead down to around 1231 MB reducing it by only 167 MB which is just one-fifth of idle mysql case. A plausible hypothesis to explain this can be that when KVMs are running with idle mysql, most of the memory pages (out of the assigned 256 MB) are zero pages (i.e., not used)

Table II
AVERAGE THROUGHPUT FOR LXC

Benchmark	Throughput(with uskm)	Throughput(without uskm)
epinions	462.3	477.5
tpcc	12.98	11.89
seats	108.87	100.23
ycsb	423.3	426.13

and hence uKSM could share these pages. But when OLTP benchmark is run, these pages are used to load database records and hence the memory content across KVMs is different. Since these zero pages are being reused, we don't see a significant increase in memory usage and hence the major contribution to shared memory comes from the kernel pages and application code (e.g mysql).

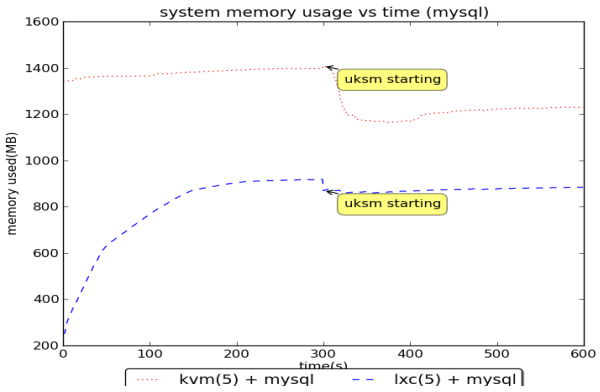

Figure 8. Memory overhead for five KVMs/LXCs running mysql with OLTP benchmark

F. OLTP throughput analysis of LXC/KVM(s) running mysql server

To identify if uKSM has any degrading effect on the throughput of OLTP benchmarks because of sharing of pages, we analyzed the OLTP logs as mentioned in Section IV-E and plotted the throughput (requests per sec) for each of the benchmarks against time (Refer Figures 9, 10, 11, 12). However, looking at the two tables (See Table II and Table III), we observe that uKSM did not degrade the performance of any of the benchmarks we ran. The throughputs for each of the benchmarks with uKSM turned on and turned off are comparable. Hence, it is safe to assume that the performance does not suffer as uKSM works in background to share memory and decrease memory utilization.

If we compare LXC with KVM in terms of througput, LXC outperforms KVM in epinions, tpcc and seats. However in the case of ycsb, KVM performs better. This is because,

Table III
AVERAGE THROUGHPUT FOR KVM

Benchmark	Throughput(with uskm)	Throughput(without uskm)
epinions	102.6	100.5
tpcc	11.82	10.62
seats	53.7	54.13
ycsb	775.5	746.5

ycsb is a read intensive benchmark, whereas epinions, tpcc and seats are write intensive benchmarks. For read intensive application, page cache and cache hits are important parameter for performance. For KVM, the memory isolation were provided by VMM, where LXC has to share the host OS page cache and hence no isolation between memory. As a result, more throughput were observed with KVM compared to LXC due to high cache hits.

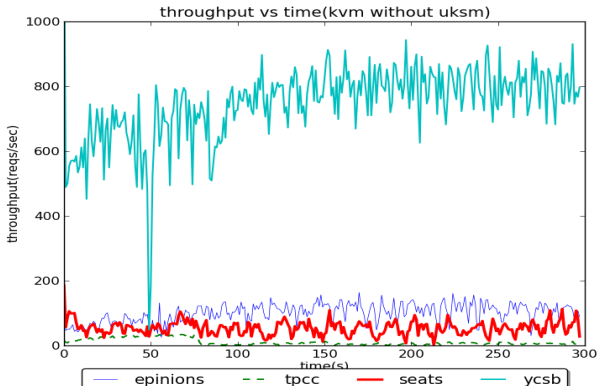

Figure 9. Throughput of various benchmarks without uKSM for KVMs

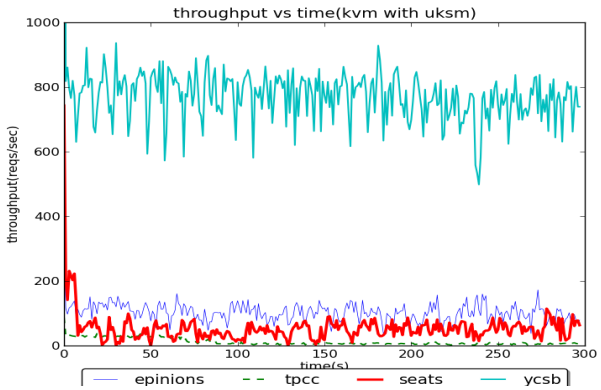

Figure 10. Throughput of various benchmarks with uKSM for KVMs

G. Boot time analysis

We measured how much time it takes for 'n' KVM/LXC(s) to boot up. We assume that a KVM/LXC has started (or is ready) if a ping to the assigned IP address

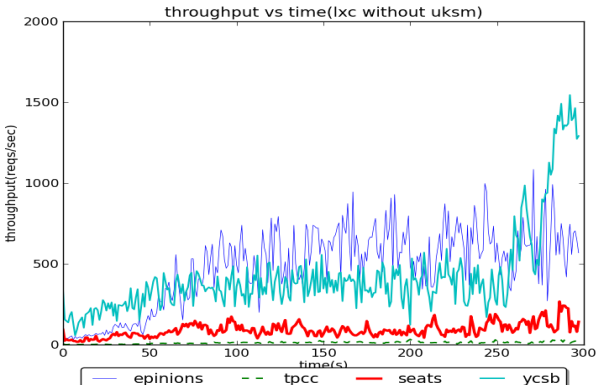

Figure 11. Throughput of various benchmarks without uKSM for LXCs

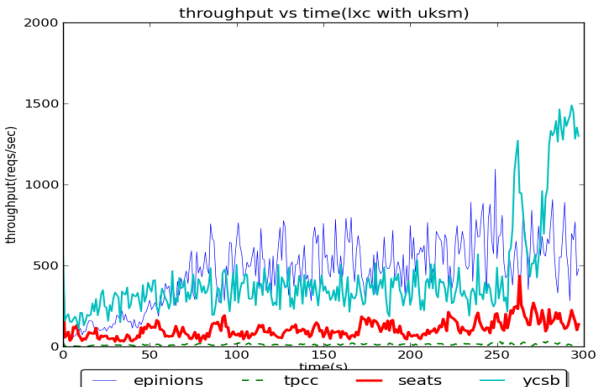

Figure 12. Throughput of various benchmarks with uKSM for LXCs

succeeds. Hence, when we are able to ping all the 'n' of them, we record it as the boot time for the 'n' KVM/LXC(s). We performed this experiment for KVM/LXC(s) running apache servers. We varied 'n' in the range : 1 to 10. Refer to Figure 13. To avoid disk utilization during boot up, we made sure that all disk block required to boot up the KVM/LXC were present in the cache. This cache setup is used in today data centers as well to decrease the boot up time. Hence, we used the similar setup.

We observed that LXC boots very fast and is ready within a second or so. Ten LXCs running apache server boot in just 1.41 seconds. On the other hand, booting KVMs is a relatively slower process. Booting a single KVM with apache server takes around 4.1 seconds (when all required disk blocks are in cache). However, the total boot time almost remains constant even when multiple KVMs are started. Ten KVMs take 5.38 seconds to boot. Since CPU is the bottleneck here and the blade server has 8 cores, the time taken to boot multiple KVMs don't vary significantly.

Figure 13 also shows the CPU usage when the LXC/KVM(s) are booting. We see that the CPU usage during boot period of LXCs is less than that of KVMs. Ten KVMs

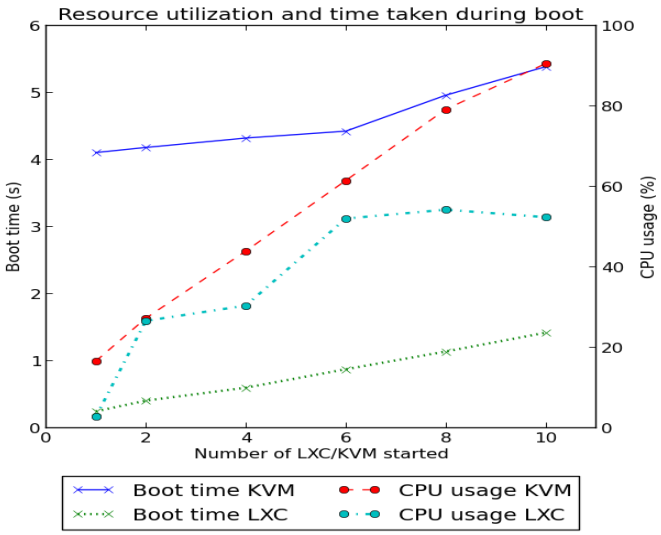

Figure 13. Booting time for KVM/LXC(s) running Apache

use around 90.42 % CPU during boot as compared to 52.23 % CPU used in case of LXCs. Hence, we can conclude that LXCs boot faster and also has CPU usage less than KVMs.

V. Conclusions

We observed that though uKSM reduces the memory footprint of KVMs significantly, but still KVMs have more memory overhead than LXCs running similar configuration. Running OLTP benchmarks as a load reduces the shareable memory in LXCs to one-third and in KVMs to one-fifth of the shareable memory in case of idle mysql server. uKSM does not degrade the performance of OLTP benchmarks in both LXCs and KVMs. KVMs have higher boot times as compared to LXCs which boot within a second or two even when ten of them are started. LXCs are also light on CPU resource during boot period. Hence, we can conclude that when the guest kernel is same as that of host, LXCs are a better choice in terms of memory and CPU usage.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177. [Online]. Available: http://doi.acm.org/10.1145/945445.945462

[2] A. Kivity, "kvm: the Linux virtual machine monitor," in *OLS '07: The 2007 Ottawa Linux Symposium*, Jul. 2007, pp. 225–230.

[3] B. Walters, "Vmware virtual platform," *Linux J.*, vol. 1999, no. 63es, Jul. 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=327906.327912

[4] "Linux Container." [Online]. Available: http://lxc.sourceforge.net

[5] "openVZ." [Online]. Available: http://www.openvz.org

[6] "Linux VServer." [Online]. Available: http://linux-vserver.org

[7] A. S. Kumar, "Virtualizing intelligent river r: A comparative study of alternative virtualization technologies," Ph.D. dissertation, Clemson University, 2013.

[8] M. J. Scheepers, "Virtualization and containerization of application infrastructure: A comparison," 2014.

[9] X. Tang, Z. Zhang, M. Wang, Y. Wang, Q. Feng, and J. Han, "Performance evaluation of light-weighted virtualization for paas in clouds," in *Algorithms and Architectures for Parallel Processing*. Springer, 2014.

[10] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013.

[11] J. Che, Y. Yu, C. Shi, and W. Lin, "A synthetical performance evaluation of openvz, xen and kvm," in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*. IEEE, 2010.

[12] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," *IBM Technology Journal*, vol. 28, p. 32, 2014.

[13] "Extending KVM with new Intel Virtualization technologyLinux Container." [Online]. Available: http://www.linux-kvm.org/wiki/images/c/c7/KvmForum2008$kdf2008_11.pdf

[14] "PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology." [Online]. Available: http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html?wapkw=sr+iov

[15] "KSM: Kernel Same Page Merging." [Online]. Available: https://www.kernel.org/doc/Documentation/vm/ksm.txt

[16] "uKSM: Kernel Dedup." [Online]. Available: http://kerneldedup.org/en/projects/uksm/

[17] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," *Communication of ACM*.

[18] C.-R. Chang, J.-J. Wu, and P. Liu, "An empirical study on memory sharing of virtual machines for server consolidation," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, May 2011, pp. 244–249.

[19] S. Barker, T. Wood, P. Shenoy, and R. Sitaraman, "An empirical study of memory sharing in virtual machines," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX, 2012, pp. 273–284. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/barker