Understanding The Network I/O Performance of Heterogenous Virtualisation in Cloud Data Centres

Technical Report

Fung Po Tso*
*Department of Computer Science, Liverpool John Moores University, L3 3AF, UK
Email: p.tso@ljmu.ac.uk

***Abstract*—With the advent of virtualisation, virtual machines (VMs) running on under-utilised physical servers can be consolidated and reassigned into fewer hosts to improve resource utilisation and reduce operational cost. Alongside a number of existing hypervisor-based virtualisation technologies such as Xen, KVM and VMWare, container-based virtualisation received considerable attention in recent years. While this type of virtualisation trades isolation for efficiency, the performance of collocating both types of virtualisation within a Cloud Data Centre remains largely unknown. In this paper, we present a performance measurement study of network I/O applications for both types of virtualisation over a Cloud environment. We focus on the performance impact of collocating applications and various virtualisation techniques in a virtualised Cloud Data Centre in terms of network throughput. We show that some types of VMs are more vulnerable to interference which leads to performance degradation of 30%. We also show that collocation of VMs of different types can improve performance by up to 25%.**

***Keywords*—Virtualisation, Hypervisor, Container, Virtualised I/O Performance, Docker, Virtualisation Overhead**

I. Introduction

A hypervisor or Virtual Machine Monitor (VMM), such as KVM, Xen and VMWare, is a popular mechanism for implementing virtualisation in a Cloud environment. Recently, another type of virtualisation technology, container-based (or OS-based) virtualisation, has attracted significant attention by the industry. Since it was released as an open-source project in March 2013, Docker [1], a Linux Container (LXC)-based virtualisation mechanism, has rapidly risen to the second most popular open-source project in cloud computing [2], and has received interest from major Cloud operators [3][4].

As their efficiency has great impact on the overall utilisation of the virtualised infrastructure, the performance of hypervisors in isolating and sharing resources has been scrutinised to a great extent. However, existing work either only examines homogeneous environments in which only one hypervisor is used, or only employ a simple setup to compare the performance of hypervisor-based VMs and containers [5][6] [7]. With the growing trend of heterogeneous Cloud environments such as increased popularity of federated clouds [8][9][10], we argue that it is very likely for Cloud operators to collocate different types of VMs to exploit their resource utilisation characteristics. For example, paravirtualised VMs, fully virtualised VMs, and containers may be collocated to exploit efficiency and performance isolation. Nevertheless, existing literature provides only limited insight for such heterogeneous collocation.

In this paper, we put the emphasis on the performance measurement and analysis of network-intensive applications with *heterogeneous virtualisation* in a heterogeneous virtualised Cloud. To gain insight and full understanding of resource sharing, isolation, and efficiency, we argue that it is important to conduct an in-depth study of applications running on *multiple heterogenous VMs* (i.e. paravirtualised VMs, fully virtualised VMs, and containers) hosted on a single physical machine. Such measurements can offer deeper understanding of the key factors for effective resource sharing amongst different applications running in the Cloud. We focus on network I/O applications in this measurement study since they constitute the primary Cloud workload, and since network utilisation is the key performance contributor to the overall cloud performance[11].

We dedicate our measurement study to answer two questions in relation to collocating heterogeneous VMs: "*How well can different combinations of VMs perform if they are put on the same physical host?*" and "*Is container virtualisation more efficient in a shared virtualised environment?*"

Through our measurement study, we show the impact of their network I/O performance by collocating multiple containers with other types of VMs. We trust the findings from out study will help Cloud operators to effectively exploit and manage their virtual environments to meet diverse requirements through running different VMs to suit their customers' needs.

In summary, the contributions of this paper are as follows:

- We performed an extensive experimental study on collocating heterogenous VMs and applications in a physical host.
- We find that a fully virtualised VM is as efficient as a paravirtualised VM in terms of network I/O applications.
- We reveal that running containers inside hypervisor-based VMs suffers significant performance impairment despite better isolation.
- We demonstrate that collocation of different VMs provides distinctive performance results. In all combinations we tested, we found that collocating a fully virtualised VM and a paravirtualised VM yields the best performance results.

The remaining of this paper is organised as follows:

Section I presents background on virtualisation technologies that inspire our performance measurement work. We describe our testbed settings, test environment and performance metrics in Section III followed by discussion and analysis of baseline results from running a single VM out of all types of VMs under test in Section IV. We present, compare and contrast more extensively the results that include running multiple homogeneous and heterogeneous VMs on the same physical host in Section V and Section VI, respectively. We survey related work in Section VII, and Section VIII concludes our paper.

II. Background

A. Virtualisation Technologies

There are a variety of VM architectures ranging from the hardware such as Intel's virtualisation Technology (or Intel VT) up the full software including hardware abstraction layer VMs such as Xen [12], KVM, VMware, system call layer VMs such as Linux VServer, OpenVZ and Linux Container (LXC).

1) Hypervisor-based virtualisation: The hypervisor is a software that provides the underlying system hardware abstraction through which multiple guest operating systems (OS) are allowed to run concurrently on a single host system. There are two main virtualisation methods [13]. The first one allows to run any kind of OS and emulates all the necessary hardware to create an impression that the guest system is running on a physical machine. We refer this type of virtualisation to as hardware assisted virtualisation and VMs on top of which are hardware virtual machines (HVMs). Second approach is to run a modified guest operating system, which is "aware" of being virtualised. The latter, called paravirtualisation, is much more efficient, but limited to some operating systems only [12]. We refer to VMs running atop as PVM.

Both full and para-virtualisation are supported by Xen [14]. To make the IO operations as fast as possible, Xen introduced also paravirtualised device drivers. Each guest domain, unprivileged domains (DomU) has the front-end drivers installed. Such drivers, provided with Xen, are communicating with the back-end drivers running on a special driver domain (Dom0). In comparison, a HVM does not have the PV drivers located within the virtual machine; instead a special daemon is start ed for each HVM Guest in Domain 0, Qemu-dm. Qemu-dm supports the Domain U HVM Guest for networking and disk access requests [15].

When a packet arriving on the physical device NIC is routed via the driver to the bridge and to the virtual interface. The back-end then notifies the front-end of a packet arrival, which is transmitted across the virtual interface to the front-end driver in the guest VM. The newest version of Xen uses the credit scheduler [16]. It assigns two parameters for each domain - weight and cap. The weight defines how much CPU time a domain gets comparing to other virtual machines. The cap parameter describes the maximum amount of CPU a domain can consume. This two parameters are then used to calculate the number of credits which determine wether a VM can be scheduled.

2) Container-based virtualisation: A container-based system provides a shared, virtualised OS image consisting of a root file system, a shared set of system libraries and executables.Each VM can be booted, shut down, and rebooted just like a regular operating system. Resources such as disk space, CPU guarantees, memory, etc. are assigned to each VM when it is created, yet often can by dynamically varied at run time. To applications and the user of a container, the VM appears just like a separate host [17].

The resource management is only allowed via cgroups, one of Linux's modern kernel features. Thus, LXC uses cgroups to define the configuration of network namespaces and CPU sharing. The process control is also accomplished by cgroups, which has function of limiting the CPU usage and isolating containers and processes. For example, a process has a weight of 512 will have twice as much CPU time than the one that has a weight of 256. Like hypervisor-based virtualisation, network access from inside container is provided by one or multiple virtual network bridges.

III. Experimental Setup

We describe in detail our testbed, methodology and performance metrics used to evaluate different combinations of tests in this section.

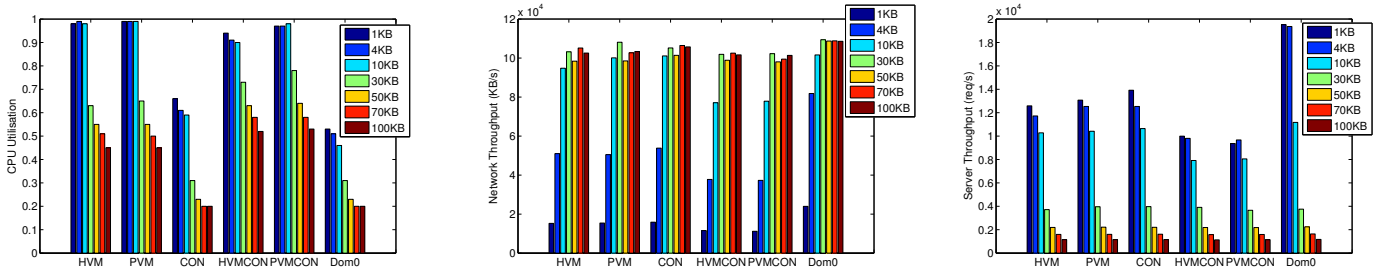
We conduct all experiments on a server and six clients each has Intel i7-3770 3.4GHz (4 cores, 8 threads) CPU and 16GB RAM. We used Ubuntu 14.04 Server distribution and Xen 4.4.0 with default credit scheduler for server and Ubuntu 14.04 desktop distribution for clients respectively. Dom0 and other guest domains have the same weight (i.e. 256) by default. This physical server hosts multiple VMs. Each VM is running Apache web server to process web requests from clients. Each client uses httperf to generates web requests for pulling web documents of size 1KB, 4KB, 10KB, 50KB, 70KB and 100KB from servers hosted within VMs. These file sizes are chosen because traffic in cloud data centre is comprised of 99% small mice flows and 1% large flows [18].

In order to recreate heterogenous VM environment, we tested the following five types of virtualisation scenarios:

- Hardware Virtual Machine (HVM): A hardware assisted virtualisation that fully virtualise the physical host.
- Paravirtualised Virtual Machine (PVM): A modified kernel exposes software interfaces, as if they are hardware interfaces, to virtual machines.
- Container (CON): A LinuX Container (LXC) based virtualisation.
- Container inside HVM (HVMCON) and PVM (PVMCON): A technique used to protect and separate containers by running them inside HVM and PVM[1].

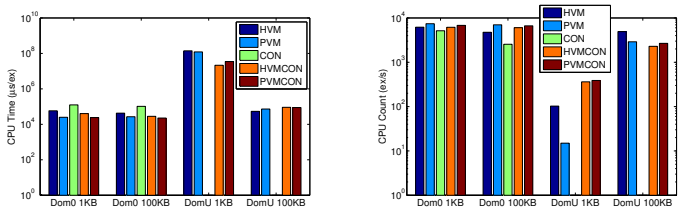
All HVMs and PVMs have a weight of 256 share so that they will have equal share to use physical CPUs. Container is also set to use 256 (out of 1024) to limit to only 25% of CPU usage as compared to two vCPUs out of eight vCPUs used for HVMs and PVMs. However, when container is running with VMs, its share is set to 1024 meaning it can use 100%

[1]Linux container faces the criticism of being less secure than VM. In order to protect and isolate container while still leveraging its usefulness for quick environment deployment, some engineers opt to run container within virtual machines [19].



(a) (b) (c)

Fig. 1: Results for single guest domain: (a) CPU utilisation, (b) Network throughput and (b) Server throughput.



(a) (b)

Fig. 2: Results for single guest domain: (a) CPU time per execution and (b) Execution count per second.

of both HVM and PVM's 2vCPUs. In this paper, we use VM interchangeably with container unless otherwise stated.

We measure and collect the following metrics, mainly from httperf [20], Xentop [21] and Xenmon [22], to benchmark VM performance:

- Server throughput (#req/sec). It measures the maximum number of HTTP requests for download specific web documents has been served by HTTP server per second.
- Network throughput (KB/sec). It measures actual intensity of network traffic has been exchanged between HTTP server and client.
- CPU Utilisation. It measures how CPU time slices are shared among VMs and how VMs use their vCPUs under given workload.
- Execution per second (#exe/sec). It measures how often a domain has been scheduled on a CPU over a period of one second.
- CPU time per execution (μs/exe). It measures the average CPU time is allocated to a running VM (including Dom0) in microsecond. Xen's Credit scheduler uses 30 ms time slices for CPU allocation.

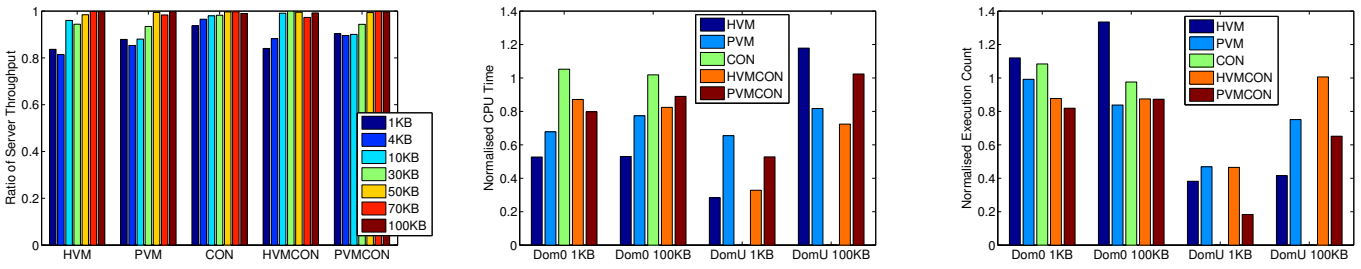
IV. Baseline Results

We consider server which has only one running guest domain or container as baseline results, assuming CPU resource is fairly shared among all vCPUs, meaning that two vCPUs guest domain will use only 25% of the physical CPU which is mapped to eight vCPUs. In this set of experiments, we first run one guest domain (thereafter in this paper we also use guest domain to include Docker container) at a time and apply one type of workload. For every type of workload, the clients initially applied 100 requests/second, and gradually increase the intensity by 100 requests/second each round until server throughput starts to fall off slightly as an increasing amount of time is spent in the kernel to handle network packets for calls that will fail eventually (due to client timeouts). We also run the experiments over Dom0, as it has the privilege to gain access to all system resource, to provide a reference point for the efficiency of running only one VM next to it. By default, Dom0 has access to all eight vCPUs.

Fig. 1 and Fig. 2 depict the results for running single guest domain in terms of CPU Usage, server and network throughput respectively. At the first glance, we can clearly observe from Fig. 1a that CPU utilisation for 1KB, 4KB and 10KB (small file) workloads are substantially higher than 30KB, 50KB, 70KB and 100KB (large file) workloads. While the same trends hold for HVMCON and PVMCON, their CPU utilisation for small workloads is 5% to 10% lower than other VMs but is 5%-15% higher in large workloads. We will see below that this phenomenon is due to both of them having smaller server throughput for small file workloads and equally high throughput for large file throughput for 1KB and 100KB workloads respectively when compared with HVM, PVM and CON. We can also see that CON (container), which runs inside Dom0, achieves comparable CPU utilisation with Dom0 for large workloads and only has 10%-15% higher for smaller workloads, and this is also because CON only uses 256 CPU share compared with that of 1024 for Dom0.

On the other hand, Fig. 1b demonstrates that network throughput for large files workload is much greater than that of small files. Fig. 1c confirms that small files workload is CPU bound whilst large files workload is network bound. Nevertheless, we also observed that container, albeit following the similar trend, consistently uses less CPU than other types of VMs by 30% on average across all workloads and still results in slightly better server and network throughput in CPU bond workload. Meanwhile, we have also seen that HVM is as efficient as PVM in all categories, contradicting to a common understanding that "fully virtualised VM is less efficient than paravirtualised VM".

More interestingly, if one wants to better protect their containers from "root break-out" [19] and run them inside HVM or PVM, i.e., HVMCON or PVMCON in our tests, we



(a) (b) (c)

Fig. 3: Results for collocating one active guest domain and three idle guest domains: (a) Normalised server throughput, (b) Normalised CPU execution time and (c) Normalised execution count.

have seen a significant drop in server throughput at 26% and 29% respectively. This implies that the tradeoff is significant as while CPU utilisation is as much as HVM and PVM. As discussed previously, container is comparably efficient as native machine while running inside Dom0. The prominent performance degradation from running container inside HVM and PVM (i.e. HVMCON and PVMCON) can be attributed to the inefficiency of running a network namespace (implemented by LXC) that provides an entire network subsystem on top of virtualised network stack. Packets coming in and out of HVMCON and PVMCON will have to traverse through an additional layer of virtual bridge between container and host VM. We however also note that during our experiments we found that installing Docker inside HVM and PVM will decrease *nf_conntrack* to a smaller value so that many TCP connection dropped and timeouted. We had to increase this value in order to carry on experiments. This implies that the underneath implementation could also affect the performance. However, discussion of Docker's implementation in greater detail is outside the scope of this paper.

Fig. 2 exhibit more details about CPU usage for 1KB and 100KB workloads respectively for all VMs. At the first glance, we can see from Fig. 2a that for both types of workloads Dom0 has received similar amount CPU time per execution, implying that both workloads have imposed similar processing demand on Dom0. However it demonstrates distinct patterns for guest VMs as they spend much greater amount of time executing small workload than large workload. Clearly, by correlating results from Fig. 2b we can see that VMs serving small file workloads have less number of scheduled execution per second as a result of having longer execution time per scheduled time. This implies that there is an efficient work aggregation of small tasks in Dom0 that results longer execution but less frequent CPU scheduling in guest domains.

In addition, Fig. 2 also reveals two interesting findings. First, with slightly higher CPU execution time but less frequent CPU schedule, container yields even better server throughput than HVM and PVM. This finding implicitly reflects good efficiency achieved by container. Second, with the same allocated CPU time but significantly lower CPU execution count, PVM achieve better server and network throughput than HVM. This is because both Dom0 and PVM are "aware" of the existence of each other, so overhead can be substantially reduced as communication only happens over special faster I/O (memory flipping) channel.

In summary, through this set of experiments, we show that container is the most efficient amongst all VMs. HVM is surprisingly as efficient as PVM when dealing with both small large file workloads although it is fully virtualised. However, running container inside HVM & PVM is the least efficient as additional layer of virtualisation brings significant overhead when processing small file workloads.

V. Multiple Homogeneous VMs

In this section, we will collocate multiple guest VM on a physical host to study how will idle and active VMs will affect neighbouring active VMs running on the same physical host.

A. Impact of Idle Domains

In this set of experiments we collocated one active domain with up to three idle domains of the same type. We are interested to understand that how would active domains be affected by idle domains. We first stressed the active domain to find maximum server throughput and repeat the same experiment by adding an extra idle domain. For the ease of discussion, we only present the results for 1KB and 100KB workload they are the most representative workloads for small and large file workloads respectively.

Fig. 3 shows the results for this set of experiments. The results are normalised with respect to results in Fig. 1 and Fig. 2 to give a better comparison.

From Fig. 3a, we can see that server's throughput for 50KB, 70KB and 100KB workload remain as high as single domain. Unlike CPU slicing, the network bandwidth is shared among all VMs through a virtual bridge. In other words, all VMs contend for network bandwidth at all times. When other idle instances are not using network bandwidth, the active one can take all, although virtualisation overhead can hurt the throughput. Nonetheless, these workloads are limited primarily by network bandwidth rather than CPU capacity. In this circumstance the server's throughput can sustain as a result of no network contention. In contrast, we can see that impairment varies from 10% to 20% in 1KB, 4KB, 10KB and 30KB workloads. This is because driver domain (Dom0) will still have to serve idles VMs even though they are in the blocked status while listening on the TCP connections.

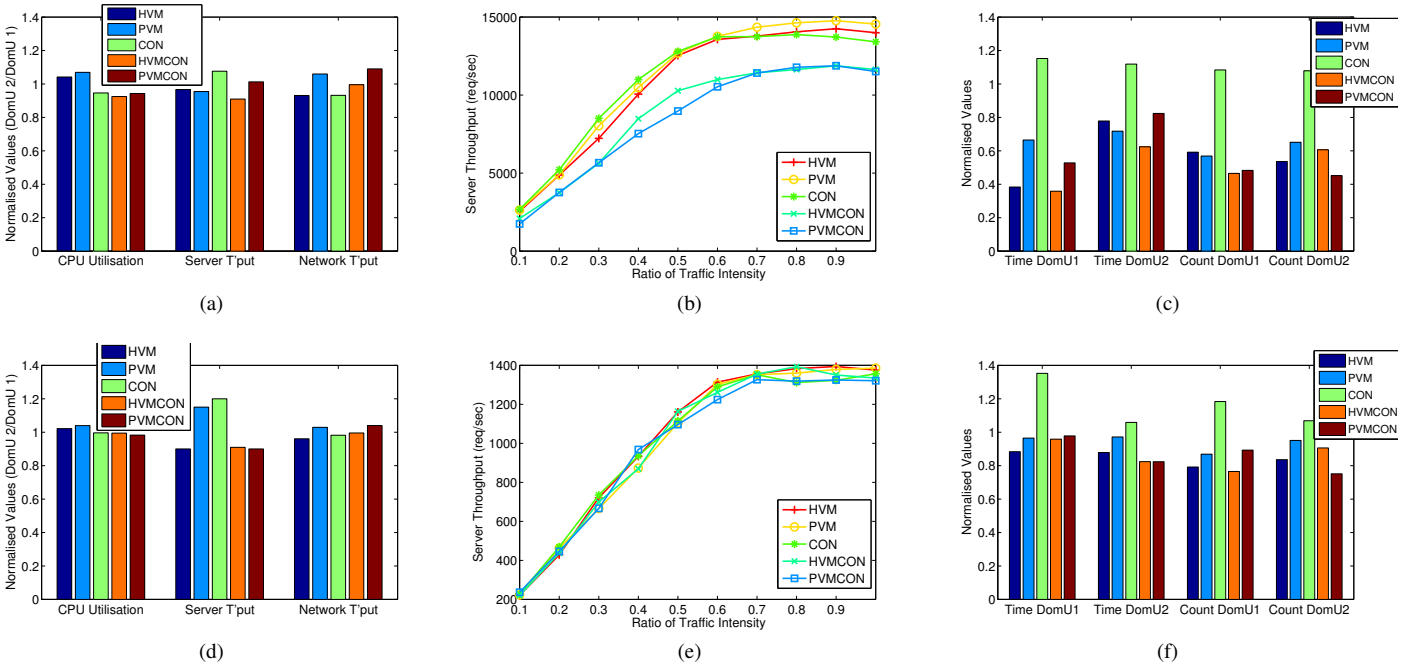


Fig. 4: Results for collocating two active guest domains with identical applications: (a) Comparison of average CPU utilisation, server throughput and network throughput for (a) 1KB workload and (d) 100KB workload; Comparison of server throughput under various traffic intensity for (b) 1KB workload and (e) 100KB workload; and Comparison of allocated CPU time and number of CPU count for (c) 1KB workload and (f) 100KB workload.

Nevertheless, the figure shows that that CON remains as the most efficient while HVMCON and PVMCON continue to be the least efficient VMs.

Next, we will examine the details of their CPU usage. Fig. 3b and Fig. 3c depict the normalised CPU execution time and count with respect to the results in Section IV for 1KB and 100KB tests.

Obviously, Fig. 3b demonstrates that in all tests both Dom0 and active DomU generally gets less CPU execution time for both 1KB and 100KB workloads as a result of I/O time-sharing and the reduction is in between 5% to 43%. Similarly in Fig. 3c, the CPU count for both Dom0 and active DomUs decreases. There are a number of factors contribute to above observations in decrease of both CPU time and count: First, it is due to the execution of timer tick for the idle guest domain and the context switch overhead. Second the processing of network packets such as address resolution protocol (ARP) packets, which causes I/O processing in DomU [5].

As oppose to decrease of CPU time and count, there are a few exceptions: First, container persistently gets more CPU time and count for both 1KB and 100KB workloads. Such behaviour is actually consistent with results discussed in previous section. In Xen, Dom0 is scheduled to use CPU as the same way as it schedules other VMs. This means that running container inside Dom0, its performance is also limited by the amount of time Dom0 is scheduled to run. In the meantime, Dom0 has to serve another three guest domain for potential I/O access. So it is scheduled to run longer and more often. Second, Dom0 for HVM has CPU execution count in between 5% to 20% greater than their counterparts in single active VM scenarios as a result of the fact that the Dom0 get remarkably smaller CPU time.

B. *Impact of Active Neighbours*

1) Identical Workload: Undoubtedly, a physical server will run multiple active VMs in order to better utilise data centre resource. The cloud operator can benefit from reduced operating expenditure. However, cloud users may worries about the performance of their applications running within sharing machine. In order to get better understanding how different types of VMs will share a physical server, we first design a set of experience to apply increasing workload intensity for all types of workload to two active VMs serving the same workload. In this set of experiments. Two identical VMs were created to run a Web server each. The clients were instructed to send HTTP request for all types of workloads with increased intensity which starts from 10% and increment by 10% in each round. The results are demonstrated in Fig. 4.

We first examine the results for 1KB workload as shown in Fig. 4a, Fig. 4b and Fig. 4c. Fig. 4a compares CPU utilisation, server and network throughput between two active VMs and demonstrates that all types of VMs can achieve fairly good fairness across all three metrics, deviating from each other only by 5% at most. Moreover, aggregate server throughput depicted in Fig. 4b reveals some interesting results: First, most pair of VMs reach their peak throughput at 80% or 90% of full workload intensity and then will fall slightly. In comparison, CON pair reach their peak throughput at 80%. Second, PVM surpasses CON as the most efficient in this test, serving 14,766 requests per second, resulting 13% higher throughput than single VM's test (in Fig. 1c). Following PVM, HVM also gain

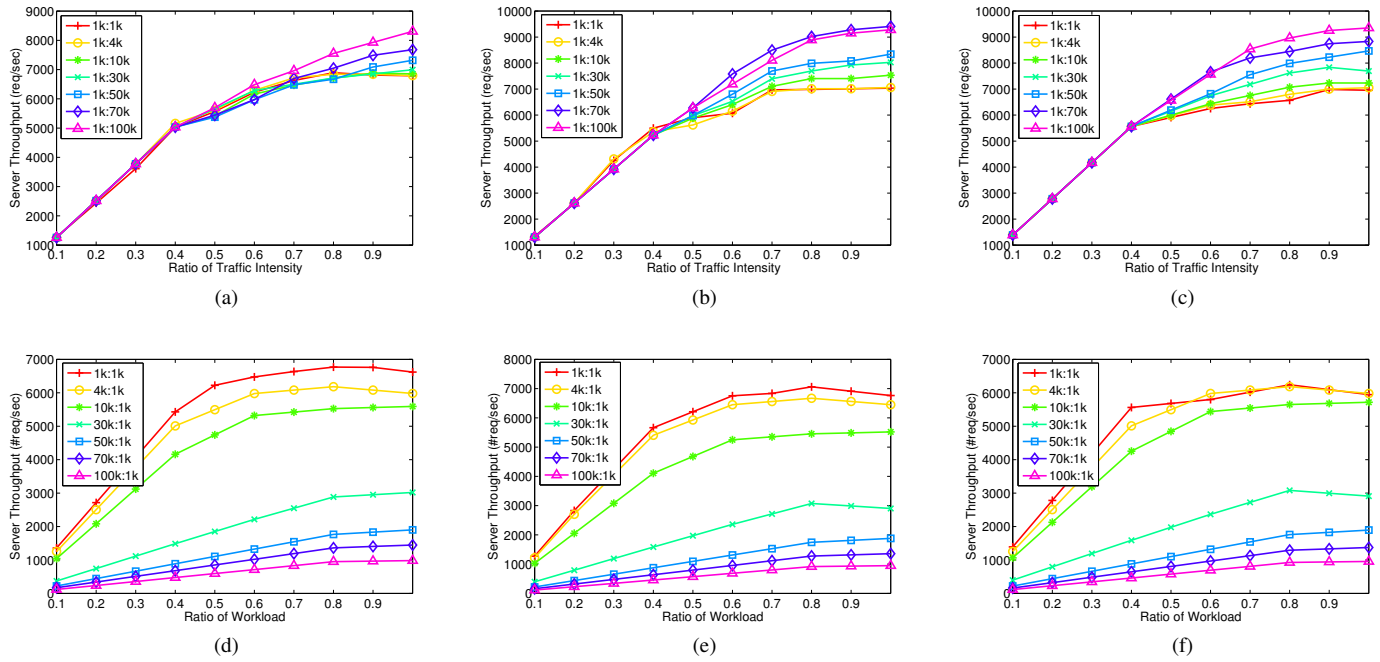


Fig. 5: Results for collocating two active VMs with different different workload: server throughput for 1KB workload for (a) HVM, (b) PVM, (c) Container, (d) Container in HVM and (e) Container in PVM

a increase 13% increase in their aggregate server throughput when compared with single VM. In comparison, CON's peak throughput falls 3% behind single or collocating with idle instances. This is because, as it is shown in Fig. 4c, both of CON instances are scheduled to run more often with higher CPU time, which in turn leads to contention in Dom0's CPU time. Hence a larger number of TCP connections (and thus HTTP requests), although arrive the servers, are kept waiting and eventually timeout. As a result, the overall performance of both CON are impaired. In other words, comparing with HVM and PVM, CON are more vulnerable to interference from collocating neighbour.

In this test, collocating two PVMs with identical workload result in the best performance. If we look at the allocated CPU time and CPU execution count in the Fig. 4c and Fig. 4f more closely, we can see that both PVM guest domains for 1KB and 100KB get nearly equal share of CPU. As a result, I/O channels between Dom0 and DomU domains are shared fairly to avoid contention on the network I/O (whose access is not scheduled).

Next, we examine the results for 100KB workload. First of all, all types of VMs achieve good fairness except CON in server's throughput, as shown in Fig. 4d and peaks their throughput at 80% or 90%, as shown in Fig/ 4e. Similar imbalance for CON in terms of CPU time and count is also observed in Fig. 4f. However, it is not reflected in server's throughput result in Fig. 4e since this type of workload is network-bound, and traffic already reaches maximum network link capacity when throughput is at peaks.

It is also worth mentioning that HVMCON and PVMCON continue to suffer from extra virtualisation overhead and thus have poorest results among all types VMs. Therefore, we omit the results from these two types of VMs in the rest of this section.

2) Different Workload: When a virtualised cloud is shared by many tenants around the world, it is more likely that VMs that belong to different tenants will be running distinctly different tasks - some are CPU bound whilst other are network bound. In this set of experiments, we will study the impact of collocating network (e.g., 100KB) and CPU (e.g., 1KB) bound task for all types of VMs. We started by dedicating one VM to serve only 1KB workload and neighbour serving varying (increasing) file size workload. We first set the ratio of traffic intensity to 10% and increase it by 10% at the beginning of next round. Fig. 5 presents our experiment results.

Fig. 5a, Fig. 5b and Fig. 5c demonstrate the server throughput for HVM, PVM and CON serving 1KB workload respectively. We used 1KB:100KB to denote that current VM is serving 1KB workload while neighbour is serving for 100KB workload. From these figures we can see that as traffic intensifies, server's throughput starts to increase. In all scenarios, 1KB:1KB hits the peaks at earliest time and either remains or starts to fall. Particularly in HVM a prominent plummet is observed. In contrast, 1KB:(50,70,100)KB curves all peaks at 100% workload intensity, while 1KB:100KB often gives best throughput (1KB:70KB in PVM), giving throughput at 9,280 requests per second. Comparing these results with Fig. 4b we can see that 1KB:100KB can achieve 28% improvement in performance.

In the meantime, Fig. 5d, Fig. 5e and Fig. 5f depict the results from their neighbouring VMs respectively. These figures exhibit that no workload suffer sudden drop in their throughput, meaning that interference from neighbour is less severe. Similarly, when serving 100KB workload (100KB:1KB), the

server reaches a throughput of 981 request per second, an improvement of 49% as compared with results in Fig. 4e

Intuitively, performance deteriorates when tasks are contenting for the same hardware resource as it will create either congestion or too frequent context switching (low yield). By collocating network and CPU bound tasks, we can minimise potential resource contention and thus improve performance.

VI. Multiple Heterogeneous VMs

An important factor to use virtualisation in the cloud is to leverage machine heterogeneity. With advance of virtualisation technologies, operators deploy a variety of VMs in their cloud to suit their customers' needs. Often, technologies used to implement these VMs are distinctively different and thus is difficult to predict their performance in advance if they're collocated and share the same physical resource. In this section, we designed a set of experiments to collocate different types of VM on the same physical host to study their operational performance.

From Sec. V-B we know that by collocating network and CPU bound workload, significant degree of performance improvement can be obtained but will see performance degradation if both run CPU or network bound workload. Therefore, in this set of experiment, we chose to collocating identical workload in order to stress test candidate heterogenous VMs. We started by sending 10% workload intensity to both servers and then gradually increase the workload to 100%.

Fig. 6 illustrates aggregate server throughput results for collocation of HVM and PVM, HVM and CON, and PVM and CON respectively. Fig. 6 shows that throughputs generally grows with growing traffic intensity. For network bound workloads, we have not observed significant improvement because they are limited by network bandwidth. Surprisingly, for CPU bound workload such as 1KB as shown in Fig. 6a, aggregate throughput for HVM-PVM (i.e. collocation of HVM and PVM) combination yields a throughput of 17,587 request per second, a remarkable improvement of 25% and 21% compared with HVM-HVM and PVM-PVM scenarios in which interference and performance degradation happens.

In comparison, collocation of HVM and CON as shown in Fig. 6b peaks at 80% workload intensity with a throughput of 14,881 requests per second, i.e. 18% behind that of HVM-PVM. Nevertheless, one might want to ask "will collocation of best performing PVM and CON achieve best performance among three candidate combinations?" Fig. 6c reveals that PVM-CON has a peak throughput of 14,887 and is also 18% behind that of HVM-PVM.

By comparing and contrasting results unveiled in Fig. 6, more surprisingly, HVM-PVM turns out to be the most efficient combination, greatly approximately the server throughput achieved by running the Web server directly inside Dom0 as shown in Fig. 1c.

We believe these results, although interesting and suprising, reflect on the performance isolation achieved by different implementation of virtualisation technology. As discussed in Section II that HVM is realised upon *qemu* library and only communicates with Xen's Dom0 via dedicate *quem-dm* daemon running inside the userspace of Dom0. Whereas PVM communicates with Dom0 via shared I/O channels. By using distinct communication channels, HVM and PVM do not interfere with each other and thus can both achieve their best performance which in turn resulting in the best aggregate results.

But then, "How would HVM and PVM interfere with CON when they are collocated?". It is true that container does not share any communication channels with HVM or PVM either, but since it is run inside and thus share CPU time of Dom0 which also provides I/O (network and storage) access support for DomU, it indirectly contends with DomU for CPU and I/O.

VII. Related Work

Recent years have seen significant research and development in virtualisation technologies. As software growing complex, it is difficult to understand and predict its performance based on simple models.

There are a number of VMM (virtual machine monitor) monitoring tools [21][22][23] developed for Xen [12]. Xentop [21] is able to displays information about the Xen system and domain in real time. We used Xentop to collect the utilisation of CPU, RAM and network of a particular domain. In comparison, Xenmon [22] provides us insights into CPU scheduling such gotten time, wait time and blocked time respectively. While the combination of Xenmon and Xentop enables us to perform many debugging or performance tracing tasks. Tools like Xenoprof [23] supports system-wide coordinated profiling in a Xen environment to obtain the distribution of hardware events such as clock cycles, instruction execution, TLB and cache misses, etc.

There is another direction of research on Xen's performance measurement. In [5] and [6], authors conducted large scale experiments in order to understand the performance interference between collocating VMs. While interesting, their works only focus on PVMs and do not provide insights into HVMs and recently proliferating LXC containers. On the contrary, our paper covers five types of VMs that captures most, if not all, of operational VM settings. Similarly, [7] identifies clusters of applications that generate certain types of performance interference through an experiment that uses range of benchmarks and real-world traffic. On other hand, despite its current popularity, there is only a handful of work on performance analysis and measurement on container, represented by [17] [24]. In [17], the authors present the design and implementation of Linux-VServer followed by an extensive performance comparison between hypervisor-based VMs and container. Whereas, in [24] the authors evaluate the performance container in HPC environment and compare it with Xen. However, their results are limited to simple tests on system I/O speed. Whereas our work studies extensively on the performance of collocation of heterogenous VMs which are the building blocks of today's cloud environment.

VIII. Conclusion

In this paper, we conducted extensive performance measurement on heterogeneous virtualised Cloud environments. Our test results have shown that containers are an efficient virtualisation technique. However, while collocating different VMs, we found that containers are more vulnerable to

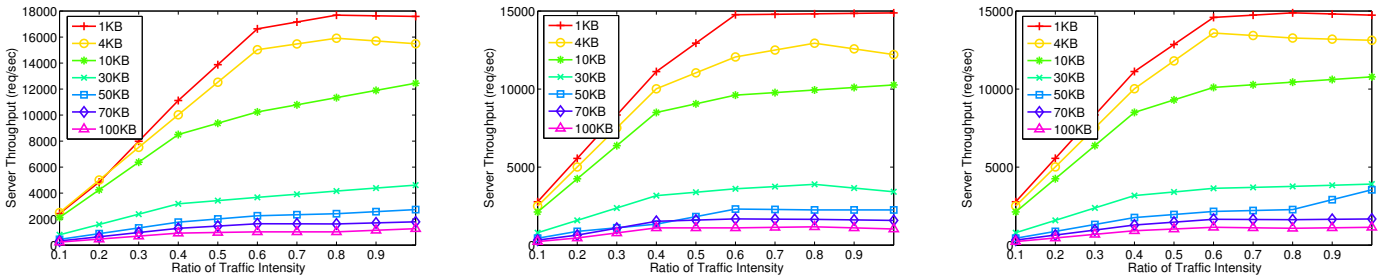


Fig. 6: Collocation of heterogeneous VMs: (a) HVM and PVM (HVM-PVM), (b) HVM and CON (PVM-CON), and (c) PVM and CON (PVM-CON)

performance interference from neighbours (i.e. HVM, PVM & CON) than hypervisor-based virtualisation. Moreover, our results have pointed out that running containers inside virtual VMs brings extra network processing overhead and hence suffers from severe performance degradation. Interestingly, we have also found that, in terms of network I/O, HVM, which is commonly believed to be less efficient than PVM [12], is as efficient as PVM under most traffic conditions. More surprisingly, we have shown that amongst all combinations of virtual environments, running HVM alongside with PVM gives the best performance results since it creates less network and CPU contention.

REFERENCES

[1] "Docker," 2014. [Online]. Available: https://www.docker.com

[2] "Openstack and docker top cloud projects," 2014. [Online]. Available: http://opensource.com/business/14/8/openstack-and-docker-top-cloud-projects

[3] "Containers on google cloud platform," 2014. [Online]. Available: https://developers.google.com/compute/docs/containers

[4] "Aws elastic beanstalk for docker," 2014. [Online]. Available: http://aws.amazon.com/blogs/aws/aws-elastic-beanstalk-for-docker/

[5] Y. Mei, L. Liu, X. Pu, and S. Sivathanu, "Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud," *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 59–66, Jul. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5558009

[6] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," *2010 IEEE 3rd International Conference on Cloud Computing*, no. Vmm, pp. 51–58, Jul. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5558012

[7] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments." in *ISPASS*, 2007, pp. 200–209.

[8] H. Fu, Z. Li, C. Wu, and X. Chu, "Core-selecting auctions for dynamically allocating heterogeneous vms in cloud computing," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, June 2014, pp. 152–159.

[9] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 826–831. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2010.46

[10] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1–4:11, July 2009.

[11] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 467–478. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626326

[12] 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," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[13] "Virtualization," *IBM Systems*, 2005. [Online]. Available: http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf

[14] P. Apparao, S. Makineni, and D. Newell, "Characterization of network processing overheads in xen," in *Proceedings of the 2nd international Workshop on Virtualization Technology in Distributed Computing*. IEEE Computer Society, 2006, p. 2.

[15] "How does xen work?" *Xen project*, 2010.

[16] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[17] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 275–287.

[18] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[19] "The docker exploit and the security of containers," 2014. [Online]. Available: https://blog.xenproject.org/2014/06/23/the-docker-exploit-and-the-security-of-containers/

[20] D. Mosberger and T. Jin, "httperfa tool for measuring web server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[21] J. Fischbach, D. Hendricks, and J. Triplett, "Xentop," *Xen builtin Utility*, 2005.

[22] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," *Hewlett-Packard Labs, Tech. Rep. HPL-2005-187*, 2005.

[23] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. ACM, 2005, pp. 13–23.

[24] 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, pp. 233–240.