The Journal of Supercomputing (2020) 76:1277–1292
https://doi.org/10.1007/s11227-018-2662-5

# mDesk: a scalable and reliable hypervisor framework for effective provisioning of resource and downtime reduction

**T. N. Sugumar**[1] · **N. Rajam Ramasamy**[2]

Published online: 31 October 2018


**Abstract**
With the advent of cloud computing and rapid integration of different architectures to form hybrid systems, the need for creation of virtual machines with the existing system is essential, to process voluminous data for a wide variety of application sets, and provide optimal solutions to their clients. This paper presents a framework for providing optimal utilization of resources through a hypervisor technique. This framework helps to reduce downtime of resources and improve machine utilization. The framework creates more than one virtual machine on a single physical machine. The performance of the partitioned machine is being examined using various signature cryptographic hash algorithms. Results show that the proposed framework is efficient and secure. The technique brings in reduced computation and supports better scalability of the virtual machines in bare metal.

**Keywords** Scalable · Downtime · Heterogeneous platform · Computation

## 1 Introduction

Traditionally, computing society uses two operating systems in a single machine either dual boot or virtualization. In dual boot methodology, two operating systems are installed in a single machine, which creates a boot menu. The programmer can access either of these operating systems while booting the machine only one at a time. Simultaneous functions of multiple OS are not allowed. The operator needs to restart the machine to access the other OS. Virtualization aims to access different operating systems in parallel without booting the machine, thereby multiple OS can be run

✉ T. N. Sugumar
sugu.agr@gmail.com

1 Department of Computing, Coimbatore Institute of Technology, Coimbatore, India

2 Department of Mechanical Engineering, Coimbatore Institute of Technology, Coimbatore, India

**Table 1** Differences between dual boot and virtualization methods

| Features | Dual boot method | Virtualization method |
|---|---|---|
| Kicking method | Rebooting process is required to execute the OS by choosing the boot menu | There is no need to reboot the process of executing the guest OS and the boot menu |
| Installation method | This method does not require third-party software to install the OS | This method requires third-party software to install a guest OS like VirtualBox, VMware, XEN and KVM |
| Performance | It cannot provide two OS running simultaneously | Guest OSes work only after host is running |
| Vulnerability | If any OS fails, it affects the entire system | If the guest OS fails, the system will not be affected. But if the host OS fails, the entire system will be affected |
| Resource allocation | Each OS uses its own system resources individually | Each guest OS uses its own system resources through the host OS |

**Table 2** Comparisons between type 1 and type 2 hypervisors

| Factor | Type-1 hypervisor | Type-2 hypervisor |
|---|---|---|
| Dependency | Entirely free from the OS for its processes | Entirely dependent on host OS for its processes |
| Physical properties | Small/thin layer hypervisor | Incurred overhead |
| Functionality | Directly runs on the bare machine, which it can monitor OS | Guest OS fully dependent on the host OS |
| Characteristics | Incorporated VMs work independently, without affecting the performance of the neighboring VMs | Disturbance in host OS or in the either of the created will affect the performance of the VMs |

simultaneously in a single machine. Table 1 shows differences between the dual boot and virtualization method of execution.

A virtualization technique aids research communities and professionals, to perform multi-tasking operations from a single platform, using different operating systems. Virtualization technology (VT) can either partition or merge the existing computing resources to one or many different operating environments. Conventional techniques currently used for performing VT are type 1 and type 2 hypervisors. Type 1 hypervisor runs directly on the bare metal and creates numerous virtual machines (VM). While type 2 hypervisors require a base operating system, the VMs are created on top of this operating system. Table 2 illustrates the comparisons between type 1 and type 2 hypervisors.

The requirement for creating a virtual machine (VM) depends on the number of physical cores present in the machine, the primary memory capacity and the capac-

ity of the secondary storage devices, other resources such as network interface card (NIC), sound card and graphics processing unit (GPU). With the advent of cloud computing technologies hybrid systems, there is a need for virtualization techniques. In the modern era, every level of business operation is moving to virtualization, cloud-based and hybrid systems, which cuts down their business costs and improves the management of information technology. Virtual CPU (vCPU) is driven by its own OS either simultaneously or successively. The simultaneous method of OS is known as bare-metal hypervisor, and the successive method of OS is known as hosted hypervisor.

The remainder of this paper is structured as follows: Sect. 2 provides a brief note on various related works published in virtualization techniques. Section 3 explains proposed approach toward virtualization describing key aspects of its design and implementation. Section 4 discusses the experimental setup for the proposed hypervisor. Section 5 uses industry standard benchmarks to evaluate the performance of the proposed hypervisor in comparison with Linux and windows families. Section 6 describes the evaluation of performance results, which are tested in various types of proposed hypervisor setup. Finally, Sect. 7 discusses future work and conclusion.

## 2 Related work

According to Kennedy [1] who performed distributed multiprocessor boot system for booting, using multiple processors in a single machine, the proposed multiprocessor interconnected architecture enabled different boot loaders to boot an OS sequentially. In the event of failure of the default processor to boot start the allocated OS, the alternative processor starts to boot its OS. The presence of multiple CPU on a bare machine enabled multiple OS to be installed, but only one OS performs the execution. A single machine carrying multiple CPUs with multiple OS was suggested. Barham et al. [2] used Xen, an x86 virtual machine monitor to support more than 100 VM instances simultaneously in a server. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP can be ported with minimal effort. Shoghet et al. [3] developed Virtuoso through the virtual network system, which was designed to provide low-level virtual machines to potential buyers, from the suppliers in a cloud environment, using the VMware GSX server architecture. This enables virtual network topology, routing and resource reservation and improves the performance of VMs.

Nanda et al. [4] designed by a Featherweight Virtual Machine (FVM) is a name space virtualization technique that virtualizes machine resources in the system call interface. This FVM proved to be an unreliable program for the real environment by permanently damaging the safe mobile code of execution and automatic uninstallation. Guangyan Zhang et al. [5] developed out-of-band virtualization system in storage area network (SAN) environment, called magic store, to obtain high persistency using the ordered writes, REDO logging and log integrity validation. It is designed to create a new log format, which provides a sliding window based on memory mapping managing, in the lost window without a lost loss. And it has been proven

in SAN environment that it can log verification and survive even during power failure. Kiyanclar et al. [6] implemented Maestro-VC architecture in a cluster management system, which executes untrusted used code in sandbox environment by the cluster administrator. It enables on-demand computing and provides controlled CPU with disk allocations using VMM for system security in a guaranteed quality of service (QoS) by the virtualized jobs.

Kinebuchi et al. [7] created flexible dynamic translator using Quick Emulator (QEMU) architecture on a portable microkernel. This model runs multiple guest operating systems simultaneously for various configurations. It reuses existing operating systems and their applications and provides higher security, reliability, flexibility and portability than existing VMMs and micro-kernels. Ito et al. [8] developed a new light-weight meso-virtualization that operated on x86 processors and two Linux OSes that successfully ran on it as guest OSes. It reduced the cost for virtualization and was suitable for embedded and ubiquitous devices. Wang et al. [9] created a Grid Virtualization Engine (GVE) for web service interface in a distributed environment on grid resources, which is a software layer that resides between the abstract layer and underlying virtualization technologies. The performance factors of scalability, availability and interoperability to the system were evaluated by GVE.

Baldin et al. [10] created Innocuous Register File Mapping (IRFM) in hybrid architecture for exchanging full and para-virtualization to support real-time applications. This IRFM offers hierarchical scheduling mechanisms, and it permits memory mapping between virtual registers and pages inside the VM manageable memory area. Thein et al. [11] developed Virtual A Machine-Based Software Rejuvenation (VMSR) framework approach using stochastic modeling in cluster environment. It is used to host multiple VMs in a single physical server. Steady-state system availability, downtime and downtime costs were derived by this model with numerical analysis, and the results were validated through Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) tool simulation. It is used as a preventive fault-tolerant technique by combining the software rejuvenation and virtualization. Yu et al. [12] presented Xen para-virtualization snapshot mechanism for virtual disks in private cloud architecture, called SNPdisk (SNP-snapshot). It helps to reduce copy on write (CoW) mapping time, amount of index data and extra I/O. It constructs special sparse tree for address mapping and tests its performance by online transaction processing applications and workload for private clouds.

Jin et al. [13] did an empirical study on the impact of server virtualization on energy efficiency from native server to virtualized servers with Xen and KVM. They show the fundamental trade-off in virtualized servers, which dictates how server consolidation should be designed and deployed to tame the explosive energy usage in green data center architecture. Kundu et al. [14] presented a VM-hosted application for cloud service providers and users that allocate resources to the VM which evaluates the use of the artificial neural network (ANN) to support vector machine (SVM) techniques. Using these techniques, the prediction of errors considerably reduced in virtualized workloads from the RUBiS and File Bench suite of benchmarks for VM sizing and placement. Yang et al. [15] presented the fault tolerance mechanism in cloud environment for the improvement in cloud VM through high availability, which it achieved by

node active standby architecture and testing. This mechanism eliminated single point of failure and allowed Infrastructure as a Service (IaaS), Application as a Service (AaaS) in the virtualization clusters efficiently.

Lee and Yu [16] introduced the virtualization introspection system (VIS) in cloud infrastructure, which protects the host and VM from attacks running on KVM-based cloud. The VIS monitors both static and dynamic VM status, the identification of VMs is attacked by the hypervisor and other VMs. Also, it detects compromised VMs by securing KVM-based cloud systems. Parallel desktop developed by apple supports both Apple and Intel architecture in a single Physical CPU (pCPU) using the hardware virtualization technology. The virtual CPU's (vCPU) resources are shared from pCPU by the hypervisor techniques using resource mapping. Partitioning of HDD is allocated top CPU and vCPU for installing Apple OS and Windows/Linux OS by the Boot Camp Assistant, which is a utility tool for multi-boot software which helps to install Microsoft Windows/Linux on an Intel-based Mac. This utility tool assists users to manage the disk partition of their HDD and starts the Windows installer (device driver) for the Apple system. Also, this tool installs the Windows control panel applet for choosing the boot OS.

Based on the above addressed literature survey, the researchers concluded their results and discussions for improving the VM performance, hypervisor comparisons and introducing new techniques into VM, which are tested by benchmark on dissimilar hypervisors with various architectures. Using this approach, this paper is proposed to create a new type 1 hypervisor to run on a desktop or server. This hypervisor is made up of up to four VMs in the same machine. Also, this paper discusses the installing of Windows and Linux OS on VMs, to test its efficiency and to use resources in VMs effectively. Very few authors have reported simultaneous execution of multiple OS in a desktop on a homogenous architecture. When the host machine runs with its OS, other guests OSs are idle. When required, the guests OS runs through the third-party software, which is run on the machine's OS, thereby the guest OS execution is initiated on the host OS. vCPU performance may be lost due to high load of host OS and un-virtualized hardware. Thus, a new hypervisor called mDesk is created to increase the performance of its pCPU and vCPU to run on more than an OS in a single machine simultaneously. This makes use of unused secondary memory and unused CPU cores.

## 3 Proposed work

With the focus of hardware virtualization, it is possible to realize that any low-level resources could be converted to numerous higher-level resources concurrently. A type 1 hypervisor is proposed and developed using open-source software, to perform VM management, resource allocation and desk space allocation for each VM, common storage area for the VMs, automatic enabling of Intel-VT (Virtualization Technology) or AMD-V (Advanced Micro Devices-Virtualization), network communications among the VM and others. The developed hypervisor can be installed directly in any 32 or 64 bit, architecture of Intel or AMD. The hypervisor automatically enables Intel-VT or AMD-V. Using this process, on completion of installation procedure, the hypervisor

can generate numerous VMs in the machine. Partition of the desk space area limits the number of VMs created to be four. Moreover, the number of physical cores, available size of RAM, available size of HDD and other resources in the physical machine also affects the creation of the VM, VM allocation must be precise to OS and the total VMs used in the physical machine should not always be higher than its performance. Each VM installs appropriate OSs and verifies the common storage area and the connection between VMs.

Cores ($\nu$) is the number of processor cores of $\nu$, $v_{\text{cpu}} \in \nu$.
Cores ($p$) $\in \mathbb{N}$, where $\mathbb{N}$ is the number of processor cores.
CPU capacity ($p$)$\in \mathbb{R}$, where $\mathbb{R}$ is the processing power per CPU core.
Capacity ($p$, $r$) $\in \mathbb{R}$ is the capacity of resource type.
$r \in \mathbb{R}$, i.e., $\mathbb{R}$ contain the resource types RAM and HDD.
The number of physical cores belongs to number of physical CPUs.
The number of virtual CPUs belongs to number of physical cores.
The capacity of CPU belongs to the processing power per CPU core.
Mapping the basic resources among the virtual CPU, resources types are RAM and HDD.
The allocation of resources to the virtual CPU belongs to the physical core of the CPU.
Let $C$ be the set of all cores $\mathbb{C}_{ij} \in C$ is core '$i$' residing in CPU '$j$'
$\text{SYS}_{\max}$, where $0 \leq j \leq \text{SYS}_{\max}$, if $j == \text{SYS}_{\max}$, all unused CPUs are utilized
$\text{SYS}_{\max} * \text{Core}_{\max}$ where $\text{Core}_{\max}$ ismaximum number of cores in any system $'j'$
if $|C| == \text{SYS}_{\max} * \text{Core}_{\max}$ where Full utilization of all cores

Let SZ be the RAM size,

$\text{SZ}_{vm}$ be the amount of memory used in RAM for deploying virtualization
$\text{SZ}_{vm} = \text{SZ} - \Delta$, where $\Delta$ is threshold(max)for bare machine

### 3.1 Analysis

Amount of memory utilized by $\mathbb{C} \leq \text{SZ}_{vm}$.
Let '$V$' be the set of all virtualization machines. $|V| \leq |C|$.
$v_i \in V$ be the virtual machine$v_i$.
$V \rightarrow C, v_k \rightarrow C_{ij}$ where $1 \leq k \leq |V|, C_{ij} \in C, 1 \leq j \leq \text{SYS}_{\max}, 1 \leq i \leq$ #cores.
Let $\text{SZ}_v^k$ be the size utilized by the virtual machine $'k'$

$$\therefore \sum_{i=1}^{|v|} \text{SZ}_v^i \leq \text{SZ}_{vm}$$

## 4 Experimental setup

The two different experimental setups established using the proposed hypervisor are default VM setup and VM extension setup. The proposed hypervisor is installed in a

bare-metal machine, whose configuration is: Intel Core i5 3rd Gen @2.3 GHz Processor, 4 GB DDR3 RAM and 500 GB of hard disk. The VMs are created by the installed hypervisor as per the requirements. The default VM setup consists of two VMs, while the extension setup has more than two VMs created in the bare machine. If "$n$" no of VMs were created, they are labeled as $vCPU_1$, $vCPU_2$, $vCPU_3$, . . . , $vCPU_n$.

### 4.1 Default VM setup

The proposed hypervisor is installed on the 64bit desktop system, to create a set of two VMs which constitute the default VM setup. The two identical VMs are created in the 64bit bare machine which are identified as mDesk1 and mDesk2, whose configuration is given below.

(a) mDesk1-1 CPU-Intel Core i5 3rd Gen 2.3 GHz, 2 GB DDR3 RAM, 100 GB HDD—Windows 7 ultimate edition 64bit version
(b) mDesk2-1 CPU-Intel Core i5 3rd Gen 2.3 GHz, 2 GB DDR3 RAM, 100 GB HDD—OpenSuSE Leap 42.364bit version

### 4.2 VM extension setup

This setup is supported to create VM up to 4 in the existing system, all of which are managed by this hypervisor. Without this hypervisor in pCPU is not possibly run more than one OS simultaneously. Therefore, the proposed hypervisor enables more than one OS execution and helps to maximize the pCPU's memory and its resources. This hypervisor performs set of operations in step by step through input parameters for each VM such as enabling hardware virtualization, number of VMs required, size of RAM allocation, partitioning with allocation of secondary memory, shared storage area, choosing the network protocol, assigning the internet protocol (IP) address and allocation of terminal for each VM. Figure 1 illustrates the experimental setup for proposed hypervisor of VM extension type.

Table 3 shows the hardware and software specifications for pCPU and vCPU in the VM extension setup. After completion of the VM extension setup process, the pCPU

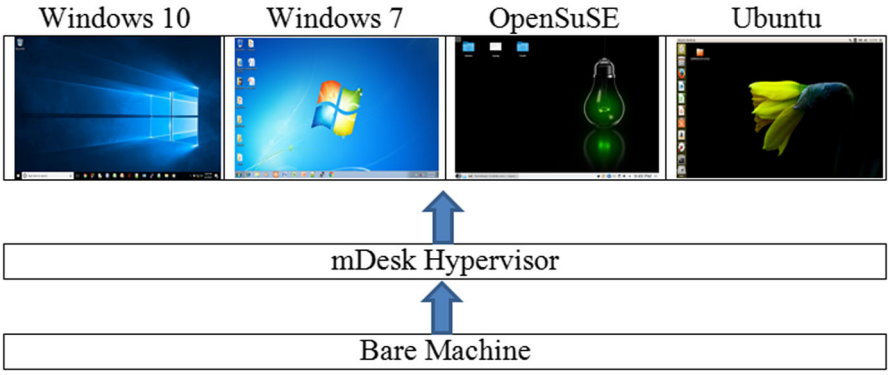


**Fig. 1** Experimental setup for proposed hypervisor installed machine

**Table 3** Hardware and software configurations for the pCPU and vCPU

| System specification | pCPU | $vCPU_1$ | $vCPU_2$ | $vCPU_3$ | $vCPU_4$ |
|---|---|---|---|---|---|
| Processor | Intel Core i5 3rd Gen 2.3 GHz—no. of core—4 | Intel Core i5 3rd Gen 2.3 GHz—no. of core—1 | Intel Core i5 3rd Gen 2.3 GHz—no. of core—1 | Intel Core i5 3rd Gen 2.3 GHz—no. of core—1 | Intel Core i5 3rd Gen 2.3 GHz—no. of core—1 |
| Memory | 8 GB | 2 GB | 2 GB | 2 GB | 2 GB |
| HDD | 500 GB | 100 GB | 100 GB | 100 GB | 100 GB |
| Operating system | mDesk Hypervisor | Windows10-x86_64 | Windows7-x86_64 | OpenSuSE Leap 42.3-x86_64 | Ubuntu16.04 LTS-x86_64 |

is powered on. Now, all of vCPUs are started to boot its OS simultaneously that are managed by the proposed hypervisor.

The VMs created were examined for the performance using standard testing procedures by quick hash software that comprises the MD5 (Message-Digest), SHA-1 (Secured Hash Algorithm), SHA-256, SHA-512 and xxHash64 algorithms as single package, which runs on both Linux and Windows platforms, to evaluate the processing time of the VMs, when executed parallel considering various combinations. Poojara et al. [17] evaluated the hypervisors performance like Xen, VMware Workstation and VirtualBox in a cloud data centers using benchmarking tools such as stress and glances. Stress written by C is a linux-based free GPLv2 (General Public License) licensed open-source software, which helps to create the OS stress and analyzing of performance. It is simple workload generator for POSIX (Portable Operating System Interface based on UNIX) systems and enforces the stress on the system by the following parameters such as memory, disk, I/O and CPU volume. Glances written by Python is to monitor linux OS from command line interface, which retrieves data from the OS. Using these tools, the performance can be measured with the use of CPU and RAM, read/write speed of disk and network. From the performance results concluded that Xen is best for the CPU, VMware is best for RAM, and disk/network VirtualBox is best. A simple combination or combination without repetition of $j$ machines from the $i$ virtual machines $\text{vCPU}_1$, $\text{vCPU}_2$, $\text{vCPU}_3 \ldots \text{vCPU}_i$ is one of the possible ways to form a set containing $j$ of the $i$ machines.

$$\text{vCPU}_{i,j} = \left(C_j^i\right) = \frac{i!}{j!\,(i-j)!}$$

## 5 Results

The execution time for the identified signature cryptographic hash algorithms, namely MD5, SHA-1, SHA-256, SHA512 and xxHash64algorithms, run on $\text{vCPU}_1$ and $\text{vCPU}_2$ created for mDesk1 and mDesk2 VMs individual is shown in Fig. 2. Table 4 illustrates the computation time of vCPUs in default VM setup, when executed parallel considering various combinations using quick hash software.

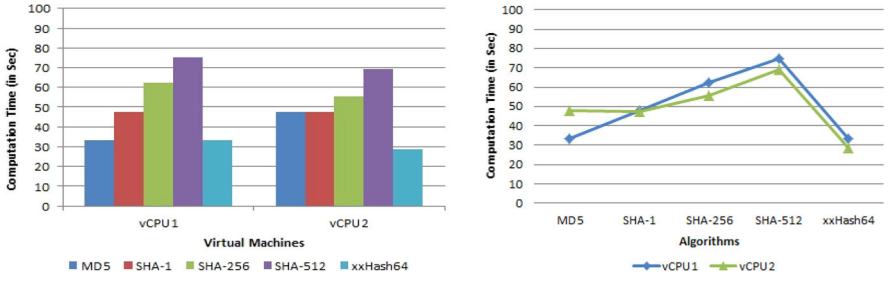


**Fig. 2** Computation time captured using quick hash on individual vCPU in default VM setup

**Table 4** Computation time of vCPUs in default VM setup

| Trial | Machines | Memory usage (MB) (%) | Quick hash algorithm's computation time in seconds | | | | |
|---|---|---|---|---|---|---|---|
| | | | MD5 | SHA-1 | SHA-256 | SHA-512 | xxHash64 |
| Windows7-Windows7-pCPU[a] | pCPU1 | 0.07 | 10 | 11 | 09 | 03 | 04 |
| 64bit-OpenSuSE Leap 42.3-pCPU[a] | pCPU2 | 0.09 | 11 | 10 | 12 | 04 | 05 |
| Each vCPU executes individually | $vCPU_1$ | 0.19 | 15 | 21 | 24 | 12 | 06 |
| | $vCPU_2$ | 0.21 | 21 | 19 | 27 | 13 | 07 |
| Two vCPUs execute simultaneously | $vCPU_{12}$ | 0.22 | 65 | 62 | 64 | 81 | 54 |
| | $vCPU_{21}$ | 0.26 | 62 | 58 | 62 | 77 | 63 |

[a]Stand-alone machines prior to loading the mDesk hypervisor

The proposed hypervisor is used to generate more than 2 VMs on the bare machine. Four VMs are created and their performance is evaluated using signature cryptographic hash algorithms. Figure 3 shows the created two virtual machines; each of them evaluates its performance of each virtual machine simultaneously in default VM setup using proposed hypervisor, which is evaluated by the above addressed algorithm execution.

Figure 4 shows the created *four* virtual machines; each of them evaluates its performance of each virtual machine individually to calculate the computation time. When a vCPU evaluates, the other *three* vCPUs are shut down on the pCPU system.

Figure 5 shows the *two* vCPUs evaluate its performances which executed its OS concurrently to calculate the computation time. When *two* vCPU evaluates, the other *two* vCPUs are shut down on the pCPU system. There are *six* possible combinations of

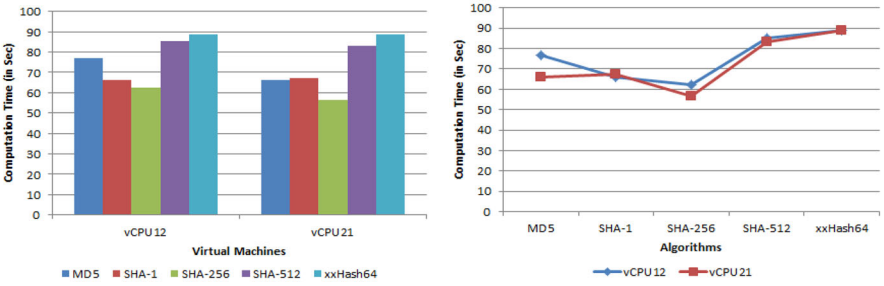


**Fig. 3** Computation time captured using quick hash on two vCPUs run concurrently in default VM setup

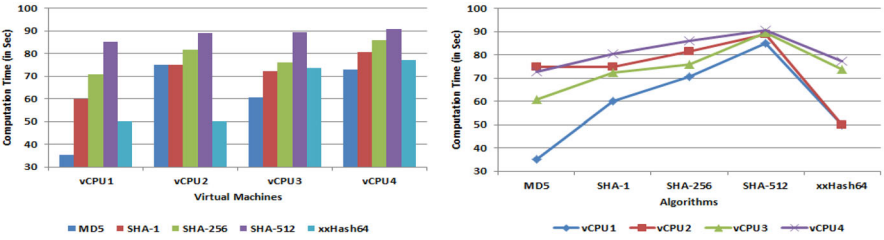


**Fig. 4** Computation time captured using quick hash on individual vCPU in VM extension setup

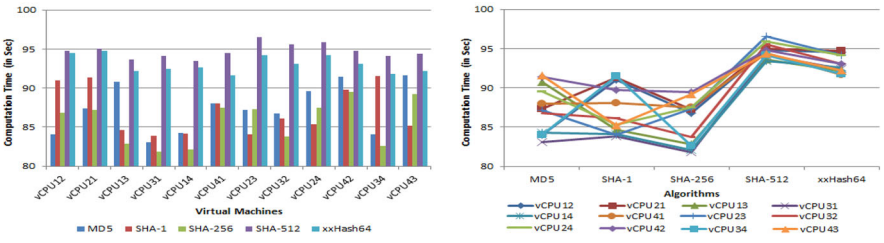


**Fig. 5** Computation time captured using quick hash on two vCPUs in VM extension setup

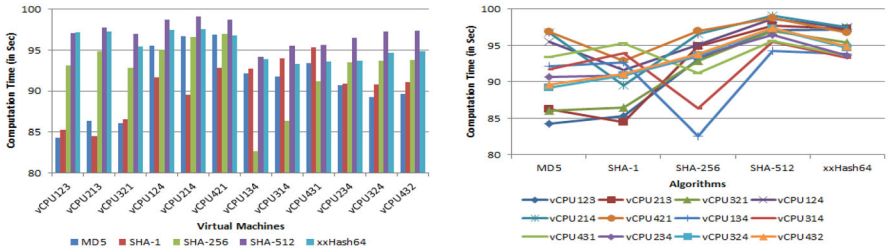


**Fig. 6** Computation time captured using quick hash on three vCPU combinations in VM extension setup

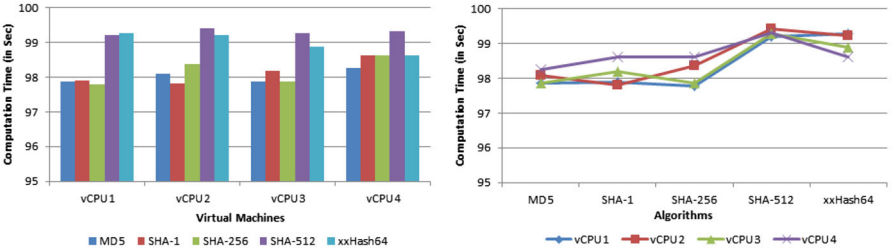


**Fig. 7** Computation time captured using quick hash on all vCPU combinations in VM extension setup

*four* virtual machines such as {$vCPU_1$, $vCPU_2$}, {$vCPU_1$, $vCPU_3$}, {$vCPU_1$, $vCPU_4$}, {$vCPU_2$, $vCPU_3$}, {$vCPU_2$, $vCPU_4$} and {$vCPU_3$, $vCPU_4$}.

Figure 6 shows that the *three* vCPUs evaluate its performances which executed its OS concurrently to calculate the computation time. When *three* vCPUs evaluate, the other vCPU is shut down on the pCPU system. There are *four* possible combinations of *four* virtual machines such as {$vCPU_1$, $vCPU_2$, $vCPU_3$}, {$vCPU_1$, $vCPU_2$, $vCPU_4$}, {$vCPU_1$, $vCPU_3$, $vCPU_4$} and {$vCPU_2$, $vCPU_3$, $vCPU_4$}.

Figure 7 shows that *all* vCPUs evaluate its performances which executed its OS concurrently to calculate the computation time. The *four* virtual machines are combined as single group {vCPU1, vCPU2, vCPU3, vCPU4}.

Table 5 illustrates the computation time of vCPUs in VM extension setup, when executed parallel considering various combinations using quick hash software.

## 6 Discussions

The computational time and the memory utilization hardware are examined to evaluate the performance of the mDesk. The default mDesk's performance with windows and OpenSuSE operating systems was compared with similar stand-alone machines equipped with the mentioned OS. Table 4 provides the computation time and the memory utilization of the systems. $vCPU_1$'s memory usage increased by 63% and $vCPU_2$ increased to 57%. When both the VMs were executed simultaneously, it was found that the memory utilization for each of the machine increased. The increase in memory utilization was not found to be uniform. $vCPU_1$'s memory utilization was found to be 13.64% while $vCPU_2$'s utilization increased to 19.23%. The variation in the memory

**Table 5** Computation time of vCPUs in VM extension setup

| Trial | Machines | Memory usage (MB) (%) | Quick hash algorithm's computation time in seconds | | | | |
|---|---|---|---|---|---|---|---|
| | | | MD5 | SHA-1 | SHA-256 | SHA-512 | xxHash64 |
| 64bit-Windows10-pCPU[a] | $pCPU_1$ | 0.09 | 11 | 10 | 12 | 4 | 4 |
| Windows7-Windows7-pCPU[a] | $pCPU_2$ | 0.07 | 10 | 11 | 9 | 3 | 4 |
| 64bit-OpenSuSE Leap 42.3-pCPU[a] | $pCPU_3$ | 0.09 | 11 | 10 | 12 | 4 | 5 |
| 64bit-Ubuntu16.04LTS-pCPU[a] | $pCPU_4$ | 0.08 | 9 | 8 | 8 | 4 | 5 |
| Each vCPU executes individually | $vCPU_1$ | 0.19 | 17 | 25 | 41 | 27 | 08 |
| | $vCPU_2$ | 0.21 | 40 | 44 | 49 | 27 | 08 |
| | $vCPU_3$ | 0.21 | 28 | 36 | 50 | 38 | 19 |
| | $vCPU_4$ | 0.18 | 33 | 41 | 57 | 43 | 22 |
| Two vCPUs execute simultaneously | $vCPU_{12}$ | 0.22 | 69 | 111 | 91 | 77 | 73 |
| | $vCPU_{21}$ | 0.26 | 87 | 115 | 94 | 80 | 76 |
| | $vCPU_{13}$ | 0.22 | 119 | 65 | 70 | 63 | 51 |
| | $vCPU_{31}$ | 0.24 | 65 | 62 | 66 | 68 | 66 |
| | $vCPU_{14}$ | 0.22 | 70 | 63 | 67 | 61 | 54 |
| | $vCPU_{41}$ | 0.25 | 75 | 67 | 64 | 72 | 60 |
| | $vCPU_{23}$ | 0.21 | 78 | 69 | 71 | 87 | 69 |
| | $vCPU_{32}$ | 0.26 | 83 | 72 | 74 | 90 | 72 |
| | $vCPU_{24}$ | 0.24 | 96 | 75 | 72 | 73 | 69 |
| | $vCPU_{42}$ | 0.26 | 105 | 78 | 76 | 77 | 72 |

**Table 5** continued

| Trial | Machines | Memory usage (MB) (%) | Quick hash algorithm's computation time in seconds | | | | |
|---|---|---|---|---|---|---|---|
| | | | MD5 | SHA-1 | SHA-256 | SHA-512 | xxHash64 |
| | $vCPU_{34}$ | 0.23 | 69 | 118 | 69 | 68 | 61 |
| | $vCPU_{43}$ | 0.26 | 107 | 54 | 74 | 71 | 64 |
| Three vCPUs execute simultaneously | $vCPU_{123}$ | 0.34 | 70 | 68 | 174 | 137 | 144 |
| | $vCPU_{213}$ | 0.37 | 73 | 71 | 176 | 131 | 146 |
| | $vCPU_{321}$ | 0.42 | 79 | 74 | 168 | 134 | 110 |
| | $vCPU_{124}$ | 0.39 | 246 | 120 | 242 | 302 | 156 |
| | $vCPU_{214}$ | 0.43 | 306 | 105 | 263 | 323 | 163 |
| | $vCPU_{421}$ | 0.44 | 293 | 112 | 268 | 321 | 154 |
| | $vCPU_{134}$ | 0.39 | 140 | 137 | 69 | 69 | 65 |
| | $vCPU_{314}$ | 0.41 | 133 | 166 | 88 | 89 | 75 |
| | $vCPU_{431}$ | 0.40 | 136 | 170 | 91 | 92 | 78 |
| | $vCPU_{234}$ | 0.43 | 107 | 121 | 138 | 85 | 63 |
| | $vCPU_{324}$ | 0.46 | 93 | 120 | 142 | 110 | 75 |
| | $vCPU_{432}$ | 0.44 | 96 | 123 | 145 | 114 | 78 |
| Four vCPUs execute simultaneously | $vCPU_{1234}$ | 0.55 | 517 | 476 | 542 | 502 | 553 |
| | $vCPU_{2134}$ | 0.58 | 523 | 502 | 552 | 513 | 518 |
| | $vCPU_{3214}$ | 0.61 | 517 | 553 | 562 | 550 | 447 |
| | $vCPU_{4321}$ | 0.57 | 516 | 578 | 583 | 587 | 362 |

[a]Stand-alone machines prior to loading the mDesk hypervisor

utilization, when both the VMs are executed simultaneously, is due to the memory requirements of the OS. The VM extension mDesk's performance with windows7, windows10, OpenSuSE and ubuntu16.04 LTS operating systems was compared with similar stand-alone machine equipped with the mentioned OS. Table 5 shows the computation time and the memory utilization of the four vCPUs' simultaneously at an instance of time on a single pCPU. The each vCPUs' average memory usage increased by 58%, concurrent execution of two vCPUs' average memory usage increased by 64.61%, concurrent execution of three vCPUs' average memory usage increased by 80.27% and concurrent execution of four vCPUs' average memory usage increased by 85.69%. When the mDesk's executing simultaneously, the computing delay time is increased gradually from the experimental results. Based on this result, the proposed hypervisor is reliable to support systems that operate dual or multiple OS in a single machine. When multiple operating systems are executed in a single machine in which resources must be upgraded.

## 7 Conclusions and future work

An attempt to develop type 1 hypervisor and evaluate performance on stand-alone machine is investigated. The capabilities of creating multiple VMs and evaluating the computation time and memory utilization are examined based on the signature cryptographic hash algorithm. As a trial, four $VM_S$ were created simultaneously, namely Windows10, Windows7, OpenSuSE Leap 42.3 and Ubuntu16.04 LTS. The evaluation results show the mDesk hypervisor is capable of working simultaneously with minimal increasing memory utilization and processing time. Utilization of CPU and memory is efficient and resource downtime is reduced. As a future work, the mDesk hypervisor will be implemented to cloud centers, which will be evaluated with necessary parameters like CPU load, memory utilization, disk read/write, network read/write and audio/video streaming using free licensed open-source softwares such as stress, glances, geek benchmark and pass mark.

## References

1. Kennedy B, Ana S (1995) Distributed multi-processor boot system for booting each processor in sequence including Watchdog timer for resetting each CPU if it fails to boot
2. Barham P et al (2003) Xen and the art of virtualization. In: SOSP'03. ACM, Bolton Landing
3. Shoykhet Alex et al (2004) Virtuoso: a system for virtual machine marketplaces. NorthWestern University, Department of Comptuer Science, Evanston
4. Nanda Susanta, Chiueh Tzi-cker (2005) A survey on virtualization technologies. Department of Computer Science, SUNY at Stony Brook, New York
5. Zhang G et al (2005) MagicStore: a new out-of-band virtualization system in SAN environments. International Federation for Information Processing, Laxenburg, pp 379–386
6. Kiyanclar N et al (2006) Maestro-VC: on-demand secure cluster computing using virtualization. National Center for Supercomputing Applications (NCSA), Univeristy of Illinois at Urbana-Champaign, Chicago
7. Kinebuchi Yuki et al (2007) Dynamic translator-based virtualization. International Federation for Information Processing, Laxenburg, pp 486–495

8. Ito M, Oikawa S (2007) Mesovirtualization: lightweight virtualization technique for embedded systems. International Federation for Information Processing, Laxenburg, pp 496–505
9. Wang L et al (2009) Grid virtualization engine: design, implementation, and evaluation. IEEE Syst J 3(4):477–488
10. Baldin D, Kerstan T (2009) Proteus, a hybrid virtualization platform for embedded systems. International Federation for Information Processing, Laxenburg, pp 185–194
11. Thein T, Sou Park J (2009) Availability analysis of application servers using software rejuvenation and virtualization. J Comput Sci Technol 24(2):339–346
12. Lei Yu et al (2011) SNPdisk: an efficient para-virtualization snapshot mechanism for virtual disks in private clouds. IEEE Netw 25(4):20–26
13. Jin Y et al (2012) Energy efficiency and server virtualization in data centers: an empirical investigation. Proceedings IEEE INFOCOM Workshops, IEEE, 2012
14. Kundu S et al (2012) Modeling virtualized applications using machine learning techniques. In: VEE'12. ACM, London
15. Yang C-T et al (2013) On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism. J Supercomput 69(3):1103–1122
16. Lee S-W, Yu F (2014) Securing KVM-based cloud systems via virtualization introspection. In: 2014 47th Hawaii International Conference on System Science. IEEE Computer Society, pp 5028–5037
17. Poojara SR, Dharwadkar NV, Ghule V (2017) Performance benchmarking of hypervisors—a case study. Indian J Sci Technol. https://doi.org/10.17485/ijst/2017/v10i44/120579