Review Article

iMedPub Journals
http://www.imedpub.com

American Journal of Computer Science and Engineering Survey

2020
Vol. 8 No.2:8

DOI: 10.36648/computer-science-engineering-survey.08.02.08.

Analysis of Wasted Computing Resources in Data Centers in terms of CPU, RAM and HDD

Robert Method Karamagi[1*] and Said Ally[2]

[1]Department of Information & Communication Technology, The Open University of Tanzania, Dar es salaam, Tanzania

[2]Department of Science, Technology and Environmental Studies, The Open University of Tanzania, Dar es salaam, Tanzania

***Corresponding author:** Robert Method Karamagi, Department of Information & Communication Technology, The Open University of Tanzania, Dar es salaam, Tanzania, E-mail: robertokaramagi@gmail.com

Citation: Karamagi RM, Ally S (2020) Analysis of Wasted Computing Resources in Data Centers in terms of CPU, RAM and HDD. Am J Comput Sci Eng Surv Vol. 8 No. 2: 08.

Abstract

Despite the vast benefits offered by the data center computing systems, the efficient analysis of the wasted computing resources is vital. In this paper we shall analyze the amount of wasted computing resources in terms of CPU, RAM and HDD.

Keywords: Virtualization; CPU: Central Processing Unit; RAM: Random Access Memory; HDD: Hard Disk Drive.

Received: September 16, 2020; **Accepted:** September 30, 2020; **Published:** October 07, 2020

Introduction

A data center contains tens of thousands of server machines located in a single warehouse to reduce operational and capital costs. Within a single data center modern application are typically deployed over multiple servers to cope with their resource requirements. The task of allocating resources to applications is referred to as resource management. In particular, resource management aims at allocating cloud resources in a way to align with application performance requirements and to reduce the operational cost of the hosted data center. However, as applications often exhibit highly variable and unpredicted workload demands, resource management remains a challenge.

A common practice to resource management has been to over-provision applications with resources to cope even with their most demanding but rare workloads. Although simple, this practice has led to substantial under-utilization of data centers since practitioners are devoting disjoint groups of server machines to a single application. At the same time, the advent of virtualization enables a highly configurable environment for application deployment. A server machine can be partitioned into multiple Virtual Machines (VMs) each providing an isolated server environment capable of hosting a single application or parts of it in a secure and resource assured manner. The allocation of resources to VM can be changed at runtime to dynamically match the virtualized application workload demands. Virtualization enables server consolidation where a single physical server can run multiple VMs while sharing its resources and running different applications within the VMs. In addition, studies have shown that reducing the frequency of VM migrations and server switches can be very beneficial for energy saving. Ultimately, server consolidation increases data center utilization and thus reduces energy consumption and operational costs.

The main challenge of server consolidation is how to dynamically adjust the allocation of VM resources so as to match the virtualized application demands, meet their Service Level Objectives (SLOs) and achieve increased server utilization. Towards this end, different autonomic resource management methods have been proposed to dynamically allocate resources across virtualized applications with diverse workload and highly fluctuating workload demands. Autonomic resource management in a virtualized environment using control-based techniques has recently gained significant attention [1].

With increasing scale and complexity of modern enterprise data centers, administrators are being forced to rethink the design of their data centers. In a traditional data center, application computation and application data are tied to specific servers and storage subsystems that are often over-provisioned to deal with workload surges and unexpected failures. Such configuration rigidity makes data centers expensive to maintain with wasted energy and floor space, low resource utilizations and significant management overheads.

Today, there is significant interest in developing more agile data centers, in which applications are loosely coupled to the underlying infrastructure and can easily share resources among themselves. Also desired is the ability to migrate an application from one set of resources to another in a non-disruptive manner. Such agility becomes key in modern cloud computing infrastructures that aim to efficiently share and manage extremely large data centers. One technology that is set to play an important role in this transformation is virtualization [2].

Virtualization

Virtualization is the process of decoupling the hardware from the operating system on a physical machine. It turns what used to be considered purely hardware into software. Put simply, you can think of virtualization as essentially a computer within a computer, implemented in software. This is true all the way down to the emulation of certain types of devices, such as sound cards, CPUs, memory, and physical storage. An instance of an operating system running in a virtualized environment is known as a virtual machine. The main idea of virtualization is to provide computing resources as pools. Depending on the needs, resources are then assigned to different applications either manually or dynamically from different pools. The scope of virtualization can vary from a single device to a large data centre and virtualization can be applied to different areas such as servers, networks, storage systems and applications.

Virtualization technology allows multiple virtual machines, with heterogeneous operating systems to run side by side and in isolation on the same physical machine. By emulating a complete hardware system, from processor to network card, each virtual machine can share a common set of hardware unaware that this hardware may also be being used by another virtual machine at the same time. The operating system running in the virtual machine sees a consistent, normalized set of hardware regardless of the actual physical hardware components. There are some other types of Virtualization technology available. For example, computer memory virtualization is software that allows a program to address a much larger amount of memory than is actually available. To accomplish this, we would generally swap units of address space back and forth as needed between a storage device and virtual memory. In computer storage management, Virtualization is the pooling of physical storage from multiple network storage devices into what appears to be a single storage device that is managed from a central console. In an environment using network virtualization, the virtual machine implements virtual network adapters on a system with a host network adapter.

The focus of server virtualization is to create virtual machines or virtual environments by using normal server hardware and a virtual machine software. Virtual machine software enables sharing physical hardware among several instances called virtual machines. Sharing is done by creating a special virtualization layer, which transforms physical hardware into virtual devices seen by virtual machines. The most visible change is the possibility to run different operating systems (OS) within the same hardware concurrently [3].

Virtualization is shown in **Figure 1** [4].

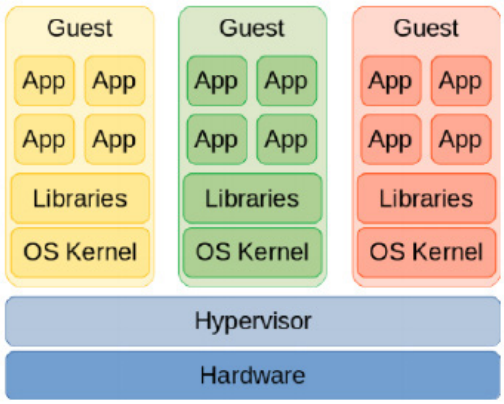

Figure 1: Virtualization.

There are three different techniques used for virtualization which mainly differ in the way they trap the privileged instructions executed by the guest kernel.

Full virtualization with binary translation: In this approach, user mode code runs directly on CPU without any translation, but the non-virtualizable instructions in the guest kernel code are translated on the fly to code which has the intended effect on the virtual hardware.

Hardware assisted full virtualization: To make virtualization simpler, hardware vendors have developed new features in the hardware to support virtualization. Intel VT-x and AMD-V are two technologies developed by Intel and AMD respectively which provide special instructions in their ISA (Instruction Set Architecture) for virtual machines and a new ring privilege level for VM. Privileged and sensitive calls are set to automatically trap to the Virtual Machine Manager (VMM), removing the need for either binary translation or para-virtualization. It also has modified Memory Management Unit (MMU) with support for multi-level page tables and tagged translation look aside buffers (TLBs).

Para-virtualization: This technique requires modification of the guest kernel. The non-virtualizable/privileged instructions in the source code of the guest kernel are replaced with hyper calls which directly call the hypervisor. The hypervisor provides hyper call interfaces for kernel operations like memory management, interrupt handling, and communication to devices. It differs from full virtualization, where unmodified guest kernel is used and the guest OS does not know that it is running in a virtualized environment [5].

Hypervisors can be distinguished into two categories: Type 1 and Type 2. Type 1 hypervisors or bare-metal hypervisors run directly and exclusively on the hardware and provide virtual machine environments to the guest operating systems. Type 1 hypervisors are commonly used for cloud computing infrastructure such as Amazon's Elastic Compute Cloud (EC2), Microsoft's Azure Cloud, VMWare, Elastic Sky X (ESX) and Xen. A Type 1 hypervisor is

also what engineers would use for virtualization on embedded systems. Type 2 hypervisors or hosted hypervisors run as an application program inside another operating system, the host, and hence share the host resources with other applications running next to them. Type 2 hypervisors are often used in desktop environments. Examples are Oracle Virtual Box or VMWare. KVM-QEMU is a popular hosted hypervisor which runs on top of the Linux operating system. Kernel-based Virtual Machine (KVM) is a kernel module providing support for hardware assisted virtualization in Linux while, Quick Emulator (QEMU) is a user space emulator. KVM uses QEMU mainly for emulating the hardware. So, both these pieces of software work together as a complete hypervisor for Linux. KVM-QEMU and Xen are open source while ESX is proprietary [4,5].

There is a correlation between energy consumption, system structures, performance and task workload in the virtual machines. When jobs are running in the virtual machines resources are employed such as CPU, memory, disk, network bandwidth which direct to energy consumption [6].

Literature Survey

Gul B et al. proposed two energy-aware Virtual Machine (VM) consolidation schemes that take into account a server's capacity in terms of CPU and RAM to reduce the overall energy consumption. The proposed schemes are compared with existing schemes using CloudSim simulator. The results show that the proposed schemes reduce the energy cost with improved Service Level Agreement (SLA).

They consider a cloud network of data centers (DCs) consisting of a large number of heterogeneous servers. Every server has its processing speed, memory, storage capacity, and energy consumption. Each individual server is represented by CPU capacity which is calculated in Million Instructions per Second (MIPS), a Random Access Memory (RAM) and a bandwidth. Servers contain local disks to host Operating Systems (OS) whereas Network Attached Storage (NAS) is used to store VMs and to enable live migration of VMs. Large number of cloud users can submit request for M number of VMs where each VM consists of its own load of CPU, memory utilization, and network transfer rate. For the management of resources, the proposed system consists of two layers: (a) a global manager and (b) a local manager as shown in **Figure 2**. Carbon emission directories are maintained for keeping energy efficiency information.

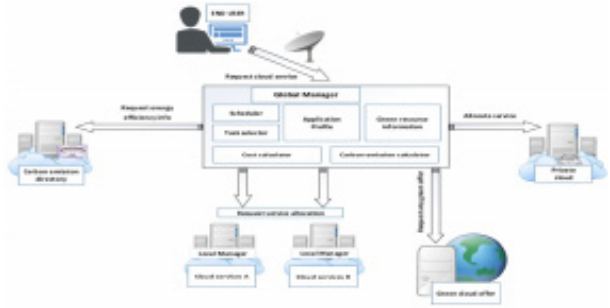

Figure 2: Green architecture for energy efficient cloud environments.

Moreover, the schemes utilize a threshold mechanism in order to keep some resources free to tackle the increased resource demands at run time. They subdivide the resource allocation problem into two components: (a) host selection and (b) VM placement. The proposed techniques take into account a PM's capacity and energy consumption while placing VMs on a server [7].

Singh A et al. describe the design of an agile data center with integrated server and storage virtualization technologies. Such data centers form a key building block for new cloud computing architectures. They also show how to leverage this integrated agility for non-disruptive load balancing in data centers across multiple resource layers-servers, switches, and storage. They propose a novel load balancing algorithm called VectorDot for handling the hierarchical and multi-dimensional resource constraints in such systems. The algorithm, inspired by the successful Toyoda method for multi-dimensional knapsacks, is the first of its kind.

They evaluate their system on a range of synthetic and real data center testbeds comprising of VMware ESX servers, IBM SAN Volume Controller, Cisco and Brocade switches. Experiments under varied conditions demonstrate the end-to-end validity of their system and the ability of VectorDot to efficiently remove overloads on server, switch and storage nodes.

They describe their system HARMONY that integrates server and storage virtualization in a real data center along with a dynamic end-to-end management layer. It tracks application computation (in the form of VMs) and application data (in the form of Virtual disks (V disks)) and continuously monitors the resource usages of servers, network switches, and storage nodes in the data center. It can also orchestrate live migrations of virtual machines and virtual disks in response to changing data center conditions [2]. **Figure 3** shows the data center testbed.

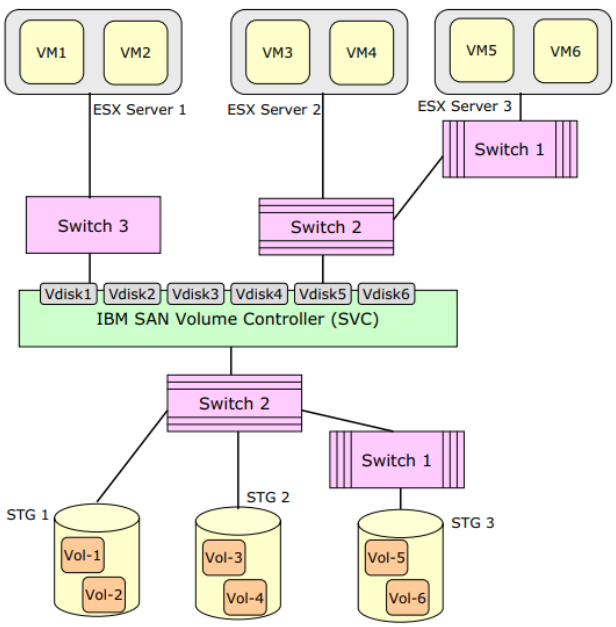

Figure 3: HARMONY testbed setup.

Gupta S and Babu MR, compare the performance of a single-core CPU, multi-core CPU and GPU using a Natural Language Processing application.

The programming over a GPU follows a Single Instruction Multiple-Data (SIMD) programming model. For efficiency, GPU processes many elements in parallel using the same program. Each element is independent from the other elements, and in the base programming model, elements cannot communicate with each other. All GPU programs must be structured in this way: many parallel elements each processed in parallel by a single program.

The Graphics Processing Units (GPU) are highly parallel rapidly gaining maturity as a powerful engine for computationally demanding applications. The GPU's performance and potential will be the future of computing systems.

General Purpose GPU (GPGPU) is a combination between hardware components and software that allows the use of a traditional GPU to perform computing tasks that are extremely demanding in terms of processing power. Traditional CPU architectures available on the market cannot satisfy the processing demands for these specific tasks, and thus the market has moved on to GPGPU in order to achieve greater efficiency is shown in **Figure 4**.

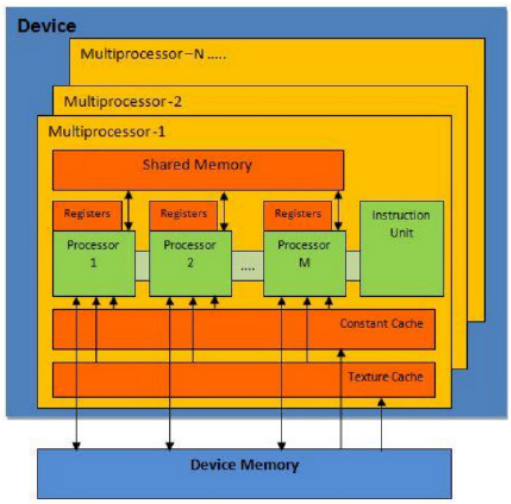

Figure 4: GPU architecture.

NVIDIA has developed Compute Unified Device Architecture (CUDA) which allows the use of the C programming language to code algorithms to execute on the GPU. CUDA enabled GPUs include data parallel cache. Besides the flexible interface for programming, it also supports memory scatter bringing more flexibilities to GPU.

CUDA provides a C-like syntax for executing on the GPU and compiles offline. CUDA exposes two levels of parallelism, data parallel and multithreading. CUDA also exposes multiple levels of memory hierarchy: per-thread registers, fast shared memory between threads in a block, board memory, and host memory. Kernels in CUDA allow the use of pointers, general load/store to memory allowing the user to scatter data from within a kernel, and synchronization between threads in a thread block. However, all of this flexibility and potential performance gain comes with the cost of requiring the user to understand more of the low-level details of the hardware, notably register usage, thread and thread block scheduling, and behaviour of access patterns through memory. All of these systems allow developers to more easily build large applications [8].

Dabbagh M et al. propose an efficient resource allocation framework for overcommitted clouds that makes great energy savings by 1) minimizing PM overloads *via* resource usage prediction, and 2) reducing the number of active PMs *via* efficient VM placement and migration. Using real Google traces collected from a cluster containing more than 12K PMs, they show that their proposed techniques outperform existing ones by minimizing migration overhead, increasing resource utilization, and reducing energy consumption.

Their proposed framework is suited for heterogeneous cloud clusters who's PMs may or may not have different resource capacities. They consider in this work two cloud resources: CPU and memory, although their framework can easily be extended to any number of resources. Their framework is made up of five modules:

- VM utilization predictor
- PM aggregator
- PM overload predictor
- Energy-aware VM migration
- PM allocation

Flowchart of the proposed framework is shown in **Figure 5**.

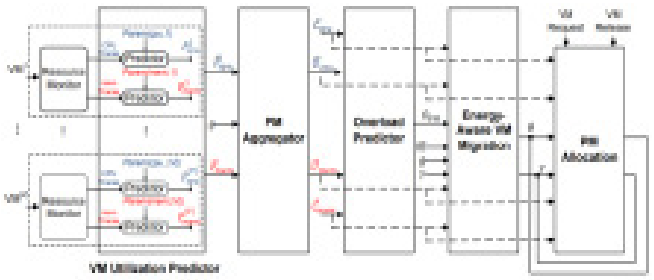

Figure 5: Flowchart of the proposed framework.

In memory overcommitment, more memory is allocated to the virtual machines (VMs) than is physically present in hardware. This is possible because hypervisors allocate memory to the virtual machines on demand. KVM-QEMU treats all the running VMs as processes of the host system and uses malloc to allocate memory for a VM's RAM. Linux uses demand paging for its processes, so a VM on boot up will allocate only the amount of memory required by it for booting up, and not its whole capacity.

On demand memory allocation in itself is not enough to make memory over commitment a viable option. There is no way for

the hypervisor to free a memory page that has been freed by the guest OS. Hence a page once allocated to a VM always remains allocated. The hypervisor should be able to reclaim free memory from the guest machines, otherwise the memory consumption of guest machines will always keep on increasing till they use up all their memory capacity. If the memory is overcommitted, all the guests trying to use their maximum capacity will lead to swapping and very poor performance.

There exists a mechanism called memory ballooning to reclaim free memory from guest machines. This is possible through a device driver that exists in guest operating system and a backend virtual device in the hypervisor which talks to that device driver. The balloon driver takes a target memory from the balloon device. If the target memory is less than the current memory of the VM, it allocates (current-target) pages from the machine and gives them back to the hypervisor. This process is called balloon inflation. If the target memory is more than the current memory, the balloon driver frees required pages from the balloon. This process is called balloon deflation. Memory ballooning is an opportunistic reclamation technique and does not guarantee reclamation. The hypervisor has limited control over the success of reclamation and the amount of memory reclaimed, as it depends on the balloon driver which is loaded inside the guest operating system [9]. Jin K and Miller EL propose the use of deduplication to both reduce the total storage required for VM disk images and increase the ability of VMs to share disk blocks. To test the effectiveness of deduplication, they conducted extensive evaluations on different sets of virtual machine disk images with different chunking strategies. Their experiments found that the amount of stored data grows very slowly after the first few virtual disk images if only the locale or software configuration is changed, with the rate of compression suffering when different versions of an operating system or different operating systems are included. They also show that fixed length chunks work well, achieving nearly the same compression rate as variable-length chunks. Finally, they show that simply identifying zero-filled blocks, even in ready-to-use virtual machine disk images available online can provide significant savings in storage [10].

Methodology

In a virtualization environment, consolidation of virtual machines is one of the techniques used to save the operational costs of data centers. Hence, virtual machine consolidation refers to the use of a physical server to accommodate more than one virtual machine for the efficient use of resources. Co-locating (consolidating) VMs reduces the number of physical servers and reduce server sprawl, a situation in which multiple, underutilized servers take up more space and consume more resources than can be justified by their workload. In addition, VM consolidation reduces the power consumption, since power consumption and the number of servers is directly related. VM consolidation can be performed in three ways: a) static, in which the virtual machine monitor (hypervisor) allocates the resource (physical resource such as memory, CPU and the likes) once and VMs will stay for long time period (such as months and years) on one physical machine. That means there will be no adjustment to the variation of workloads; b) Semi-static, in which VMs are place based on daily or weekly bases; c) Dynamic, by adjusting depending on the workload characteristics (Peak and off-peak utilization of resources) and make adjustment in hours and needs run-time placement algorithms. Dynamic VM consolidation helps in the efficient use of data centers. In order to consolidate VMs there are several processes that must be undertaken. These are VM selection, VM placement and VM migration.

VM selection is one of the challenges of VM consolidation process. It deals with migrating VMs until the physical machine is considered to be not overloaded. In VM selection process there are several policies to be followed for effective accomplishment of the process. These policies are: a) The minimum Migration Time Policy; b) The Maximum Correlation Policy; c) The Random Choice Policy and; d) Highest Potential Growth (HPG). Moreover, it is also described as: a) Local Regression b) Interquartile Range c) Median Absolute deviation.

The process of selecting the most suitable host for the virtual machine, when a virtual machine is deployed on a host, is known as virtual machine placement, or simply placement. During placement, hosts are rated based on the virtual machine's hardware and resource requirements and the anticipated usage of resources. Host ratings also take into consideration the placement goal either resource maximization on individual hosts or load balancing among hosts. The administrator selects a host for the virtual machine based on the host ratings.

Migration of VM's can be accomplished by two methods: offline migration, which has downtime because of suspend and resume of operation and; live migration, which is widely used in cloud computing and uses copying before migrating to avoid downtime. One of the most remarkable features of virtualization is live migration of VMs. In live migration active VM is transferred from one physical machine to other keeping the current working status of a VM while running. Such actions are a de facto in KVM and Xen. Live migration of virtual machines is shown in **Figure 6**.

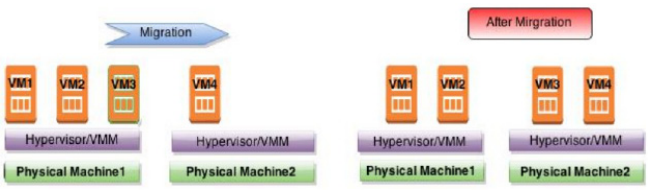

Figure 6: Live migration of virtual machines.

Fault Tolerant Migration migrates the VMs even-if system failure occurs during migration. It was assumed to minimize performance degradation of applications and improve availability.

Load Balancing Migration distributes load across the physical servers to improve the scalability of physical servers. It helps in minimizing the resource consumption, implementation of fail-over, enhancing scalability, avoiding bottlenecks and over provisioning of resources etc.

The Energy Efficient Migration conserves the energy of servers by optimum resource utilization, since the power consumption of

data center is mainly based on the utilization of the servers and their cooling systems [11].

Micro services are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services. The micro service architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

In micro services pattern, a software system is comprised of a number of independently deployable services, each with a limited scope of responsibility. Less frequently, micro services may rely on lower level services. Here each micro service can be developed, managed and scaled independently throughout its life-cycle. A well implemented micro services architecture also ensures that the overall system is able to gracefully degrade its functionality when one or more services are offline.

Containerization is also known as Operating System Virtualization. As the name implies, the abstraction is the operating system itself, instead of the platform. It is an approach to virtualization in which the virtualization layer runs as an application within the Operating System (OS). Here, the operating systems kernel runs on the hardware with several isolated guest Virtual Machines (VMs) installed above it. The isolated guest virtual machines are called containers.

Here the operating system provides a set of user-spaces that are isolated from one another, but offers the abstraction necessary such that applications believe that they are part of the singular user-space on the host. With container-based virtualization, the overhead associated with having each guest run a completely installed operating system is not there. Here there is just one operating system taking care of hardware calls, thus it improves performance. However, each guest must use the same operating system the host uses, which results in restriction for user [12]. Containerization is shown in **Figure 7** [4].

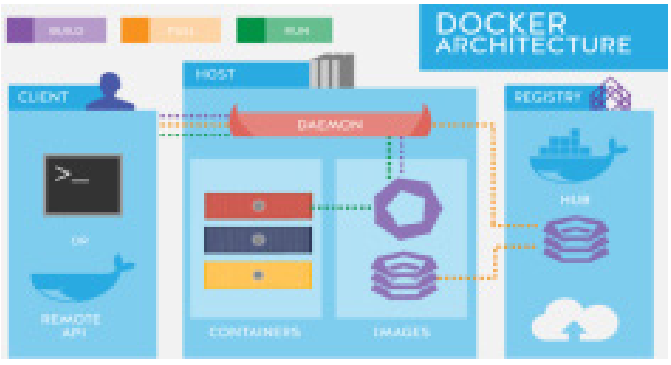

Figure 7: Containerization.

Kubernetes is a framework designed to manage containerized workloads on clusters. The basic building block in Kubernetes is a pod. A pod encapsulates one or more tightly coupled containers that are co-located and share the same set of resources. Pods also encapsulate storage resources, a network IP, and a set of options that govern how the pod's container(s) should run. A pod is designed to run a single instance of an application; in this way multiple pods can be used to scale an application horizontally for example. The amount of CPU, memory, and ephemeral storage a container needs can be specified when creating a pod. This information can then be used by the scheduler to make decisions on pod placement. These computer resources can be specified both as a requested amount and as a limit on the amount the container is allowed to consumer.

The default Kubernetes scheduler ensures that the total amount of compute resource requests of all pods placed in a node does not exceed the capacity of the node. This even if the actual resource consumption is very low. The reason behind this is to protect applications against a resource shortage on a node when resource usage later increases (e.g., during a daily peak). If a container exceeds its memory limit, it may be terminated and may be later restarted. If it exceeds its memory request, it may be terminated when the node runs out of memory. Regarding the CPU usage, containers may or may not be allowed to exceed their limits for periods of time, but they will not be killed for this. On the other hand, containers and pods that exceed their storage limit will be evicted [13].

Architecture for deploying applications to Kubernetes clusters running in a public cloud is shown in **Figure 8** [14].

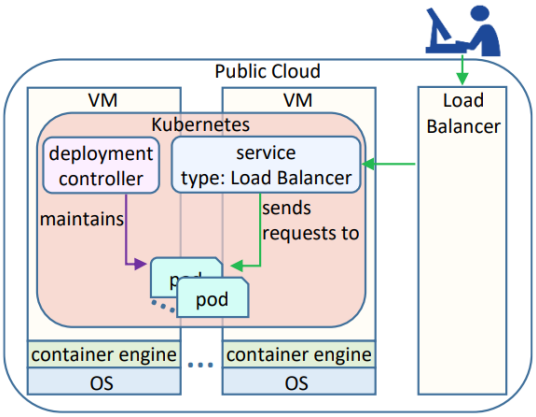

Figure 8: Architecture for deploying applications to Kubernetes clusters running in a public cloud.

The Docker platform was released in March 2013 as an open-source project based on Linux containers (LXC) and a year later, the environment was moved from LXC to lib container. Docker is based on the principles of containerization, allowing for an easy deployment of applications within software containers as a result of its innovative and unique architecture. Docker implements certain features that were missing from OS-level virtualization. It bundles the application and all its dependencies into a single object, which can then be executed in another Docker enabled machine. This assures an identical execution environment regardless of the underlying hardware or OS. The creation of applications in Docker is firmly rooted in the concept of versioning. Modifications of an application are committed as deltas, which allows roll backs to be supported and the differences to previous application versions to be inspected. This is an exceptional method of providing a reliable environment

for developers. Furthermore, Docker promotes the concept of reusability, since any object that is developed can be re-used and serve as a "base image" to create some other component. Another essential aspect of Docker is that it provides developers with a tool to automatically build a container from their source code.

The main difference between a Docker container and a VM is that while each VM has its own OS, dependencies and applications running within it, a Docker container can share an OS image across multiple containers. In essence, a container only holds the dependencies and applications that have to be run within them. For example, assuming a group of containers were making use of the same OS image, the OS would be common to all containers and not be duplicated contrary to the case of a VM topology.

Docker has become the flagship in the containerization technology arena since its release. This open-source project has gained much notoriety in the field of cloud computing, where major cloud platforms and companies (e.g., Google, IBM, Microsoft, AWS, Rackspace, RedHat, VMware) are backing it up. These companies are integrating Docker into their own infrastructures and they are collaborating in Docker's development. Recently, a few alternatives to Docker have cropped up, such as Rocket, Flockport and Spoonium [15]. **Figure 9** shows Docker Architecture [16]. An adequate monitoring of the pool of resources is an essential aspect of a cloud computing infrastructure. The monitoring of resources leads to improved scalability, better placement of resources, failure detection and prevention, and maintenance of architectural consistency, among others. This is relevant for VMs, and it is just as applicable to OS-level virtualization [15].

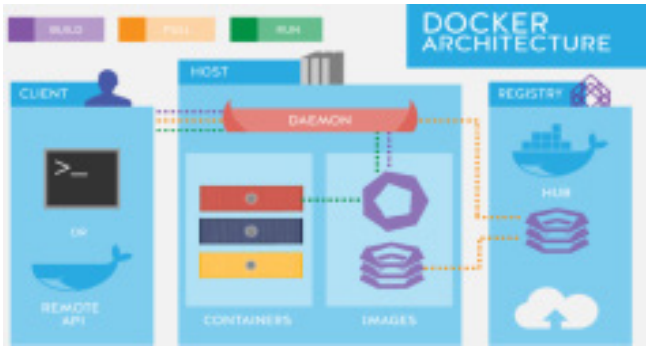

Figure 9: Docker architecture.

Storage efficiency is the process of storing and managing data while consuming the least space possible and ensuring optimal performance. This means solving problems while consuming less storage.

For many years, Storage Area Networks (SAN) and Network Attached Storage (NAS) were at the core of different parts of the data center. SANs are primarily used to make storage devices, such as disk arrays accessible to servers so that the devices appear like locally attached devices to the operating system. NAS was introduced to the market in the 1990's to accommodate the growing demand for file sharing *via* the IP network.

Thin provisioning is the act of using virtualization technology to give the appearance of more physical resource than is actually available. Traditional block storage requires all blocks to be assigned up front. This means large pools of storage are allocated to servers and remain unused. Thin provisioning allows disk to be added as needed. This means the initial acquisition costs can be reduced and further, because of the price of disk goes down over time, the long-term costs are also reduced as compared to tradition arrays that utilize thick provisioning.

Deduplication is a technology that can be used to reduce the amount of storage required for a set of files by identifying duplicate "chunks" of data in a set of files and storing only one copy of each chunk [10].

It is the method of reducing storage needs by eliminating duplicate data. Only one instance of the data resides on the disk and duplicate or redundant data is replaced with a pointer to the original.

Most think of single instance storage when it comes to dedupe but focusing there would only be part of the equation. To illustrate single instance storage or file level deduplication think of email. A user sends a spreadsheet to 10 different people. Those people all save the file on the file share and we have 10 copies in both the email system and the file server. All copies are exact copies, even down to the file name, so file level deduplication stores the file only once and replaces the duplicate copies with pointers to the original.

A more advanced and useful form of deduplication is block level or sub file level deduplication. Block deduplication looks within a file and saves unique iterations of each block. Think of a PowerPoint with a common corporate format. There are hundreds of presentations on the file share, all with the same slide design and graphics but each with their own unique content. Block level deduplication removes all the duplicate pieces of each and replaces with a pointer to the original.

An even more powerful example is block level deduplication within a virtualized server environment. Imagine a virtualized server environment with 100 VMs all with MS Windows Server 2008 R2 as the operating system. Let's say that operating system is 8 GB for each instance. That is 800 GB of operating system data, each basically a duplicate of the other. Block level deduplication replaces each block of duplicate data with a pointer and reduces the data footprint just associated to the operating system by 90% [17].

Findings

Gul B et al. results show that proposed energy-aware and SLA-aware schemes outperform the existing energy-aware and SLA-aware schemes. Improvement in results is due to improving energy efficiency which is made possible by decreasing the number of active servers. Moreover, proposed schemes monitor resource utilization to gather updated information while the existing schemes consider only maximum energy consumption for initial host selection. In their study, capacity of CPU is considered during VM placement. The server that provides maximum RAM and CPU capacities per watt power is selected

for VM placement. In this way, energy consumption is improved as compared to existing Cloud Resilience for Windows (CREW). For instance, Maximum Capacity and Power Technique (MaxCap) consumes 37% less energy than CREW and Remaining Capacity and Power (RemCap) consumes 32% less energy than CREW. The proposed SLA-aware version SMaxCap consumes 35% less energy than SCREW and SRemCap consumes 31% less energy than the SLA-aware versions SCREW [7].

Figures 10 and 11 presents the comparison of energy consumption for various PlanetLab workloads. Comparisons of their proposed energy-aware schemes (MaxCap and RemCap) and SLA-aware schemes (SMaxCap and SRemCap) are performed with existing energy-aware and SLA-aware techniques, namely: CREW and SCREW, respectively [7].

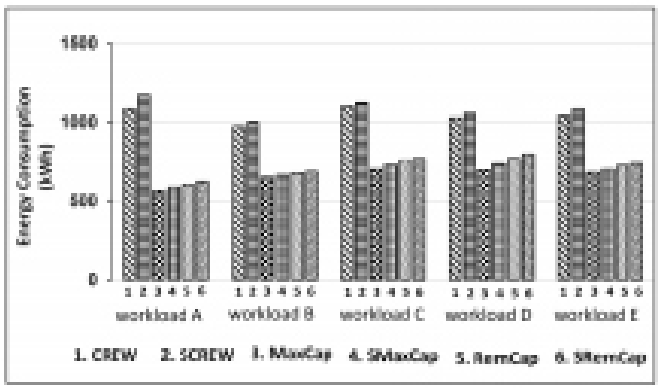

Figure 10: Energy consumption by various energy-aware and SLA aware technique.

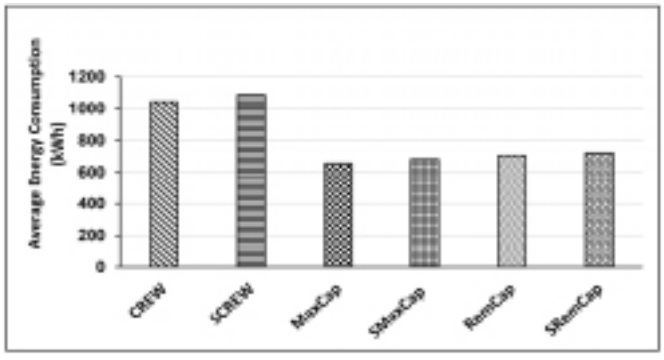

Figure 11: Consolidated average energy consumption on different workloads.

Singh A et al. for end-to-end validation of HARMONY in a real data center setup, created four scenarios in their testbed that caused overloads on multiple dimensions of servers, storage and switches. When any of the overloads are detected by HARMONY Performance Manager, it uses VectorDot to determine virtual resources that can be migrated and their destinations. It then uses the HARMONY Virtualization Orchestrator to execute suggested server and/or storage migrations by VectorDot. The description of all resources including virtual machines to host and storage mappings are shown in **Figure 12** [2].

Resource	Description		
	Server	CPU	Memory
Host Servers (VMWare ESX 3.0.1)	Server-1	6 GHz (2x3GHz)	2.6 GB
	Server-2	3 GHz (1x3GHz)	4 GB
	Server-3	4 GHz (2x2GHz)	4 GB
	Volume	Physical Controller	
Storage Volumes (20 GB, RAID 5)	Vol-1, Vol-2	STG-1	
	Vol-3, Vol-4	STG-2	
	Vol-5, Vol-6	STG-3	
Virtual Storage	Vdisk 1-6	Vol 1-6 (resp.)	
		Server	Storage
Virtual Machines (3 GHz CPU, 1.2 GB RAM, RedHat EL4.0)	VM-1	Server-3	Vdisk5
	VM-2	Server-3	Vdisk3
	VM-3	Server-2	Vdisk4
	VM-4	Server-2	Vdisk2
	VM-5	Server-1	Vdisk1
	VM-6	Server-1	Vdisk6

Figure 12: Testbed resource description.

As a first scenario, they overloaded Server-2 on the CPU dimension by creating a high CPU workload on VM-4. **Figure 13** shows the CPU and memory utilizations for all three servers with elapsed time.

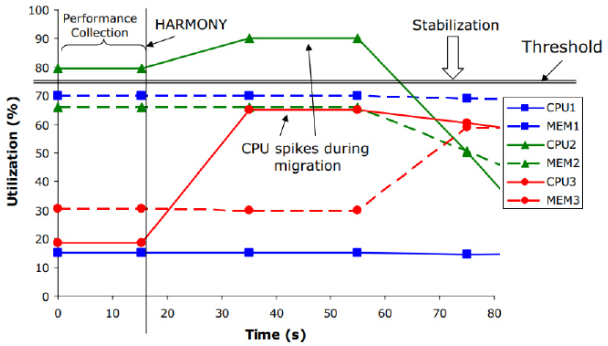

Figure 13: Single server overload resolution.

In the second scenario they created a CPU overload on Server-2 and a memory overload on Server-1 by increasing usages of VM-4 and VM-6 respectively. **Figure 14** shows utilizations with elapsed time.

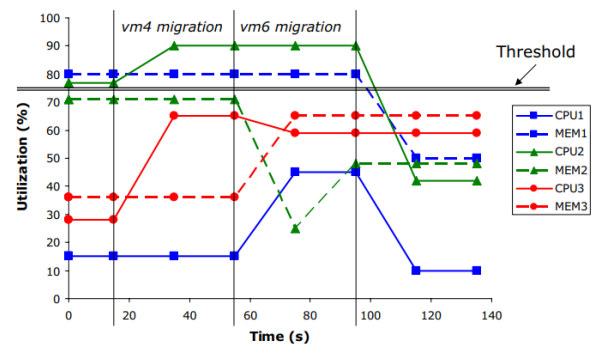

Figure 14: Multiple server overload resolution.

In the third scenario, they generated a high I/O workload on VM-5 which accesses Vdisk1 (from Vol-1 on STG-1). They also added a

memory workload to create an overload on VM-5's host server Server-1. To show utilizations for multiple resource types, they tabulated the CPU and memory utilization of servers, and storage I/O utilization of storage controllers in **Figure 15**.

	Servers (%) CPU, Mem	Storage % I/O
Initial Configuration	Server-1: 49.3, **82.4** Server-2: 62.7, 58.9 Server-3: 39.9, 47.5	STG-1: **77.8** STG-2: 9.8 STG-3: 54.9
After VM-5 Migration	Server-1: 33.3, 54.8 Server-2: 59.1, 58.3 Server-3: 67.3, 71.8	"
After Vol-5 migration	"	STG-1: 59.1 STG-2: 26.5 STG-3: 56.9

Figure 15: Integrated server and storage overload resolution.

In the fourth scenario, they fixed the capacity of Switch-2 to 50 MBps. They then generated an I/O workload on VM-4 (that uses Vdisk2 virtualized from Vol-2 on STG1) at the rate of 20 MBps. Since Switch-2 receives I/O from both Server-2 and the virtualization appliance (it is both above and under the appliance) **(Figure 3)** the workload generates 40 MBps on Switch-2. **Figure 16** shows the actual performance monitor screenshot from the switch management interface (Y-axis on log scale).

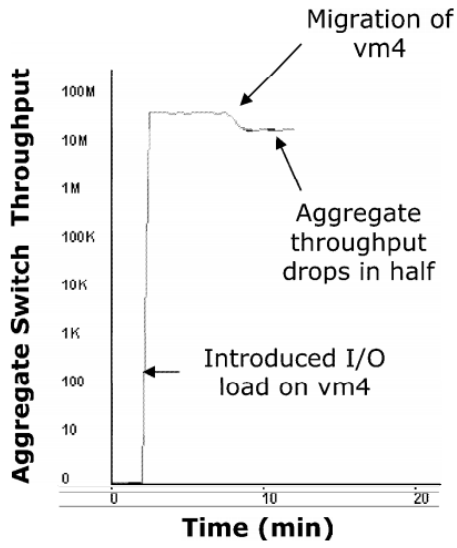

Figure 16: Switch overloads (Y-axis on log scale).

Their validation experiments demonstrate the feasibility of their integrated load balancing approach in a real deployment [2].

Gupta S and Babu MR, for their evaluation, in comparing the performance of GPU and CPU (single-core and multi-core) using an NLP application, used the following platforms.

- NVIDIA GeForce G210M 1024MB (800MHz GDDR3, 16 cores)
- CUDA version 2.1
- Open MP
- Intel Core 2 Duo CPU P8600 (2.40 GHz, 2CPUs)
- Intel C++ Compiler 10.0

As per their algorithm used, for some particular size of file (in terms of number of words in a file) their system processes the input file (performs lexical analysis and shallow parsing) and finally provides the number of matches and number of parts of speech in the provided input file [8]. The graph generated on some of the data generated on the implementation of algorithm used is displayed in **Figure 17** [8].

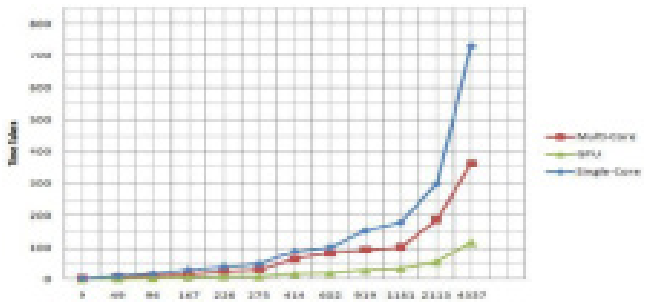

Figure 17: Performance comparison graph.

Dabbagh M et al. experiments presented in this section are based on real traces of the VM requests submitted to a Google cluster that is made up of more than 12K PMs. Since the size of the traces is huge, they limit their analysis to a chunk spanning a 24 hours period.

Resource utilization over one-day snapshot of Google traces is shown in **Figure 18**.

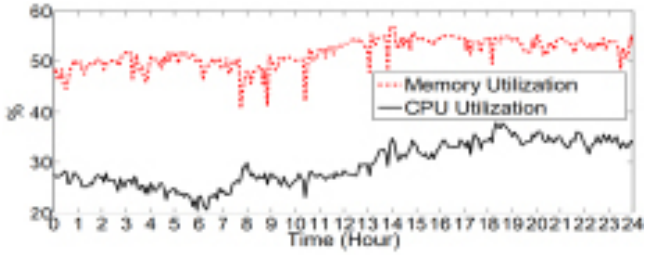

Figure 18: Resource utilization over one-day snapshot of Google traces.

Number of predicted and unpredicted overloads over time is shown in **Figure 19**.

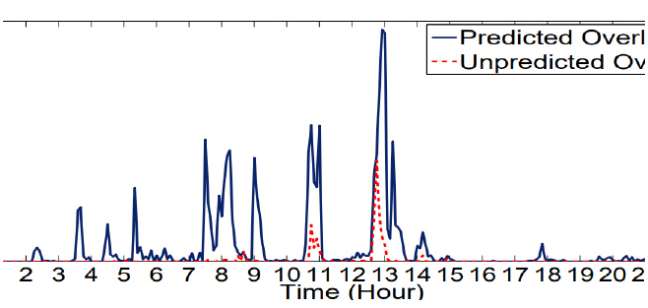

Figure 19: Number of predicted and unpredicted overloads over time.

Figure 20 plots the total migration energy overhead (including both VM moving and PM switching overheads) incurred by the

migration decisions of their proposed heuristic to avoid/handle the overloads reported in Figure 19 along with the total energy overhead associated with the migration decisions of the two existing heuristics: Largest First and Sandpiper. Both of these heuristics handle multi-dimensional resources by considering the product metrics, and both select the PM with the largest slack as a destination for each migrated VM.

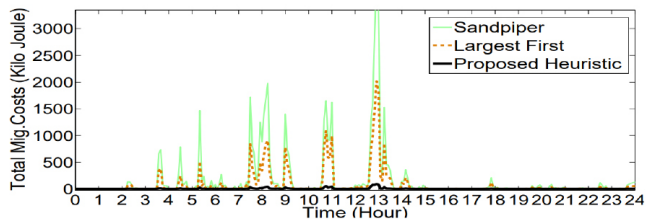

Figure 20: Total migration energy overhead under each of the three heuristics.

Since the energy consumed by ON PMs constitutes a significant amount, they analyze in **Figure 21** the number of ON PMs when running our framework on the Google traces under each of the three studied migration heuristics.

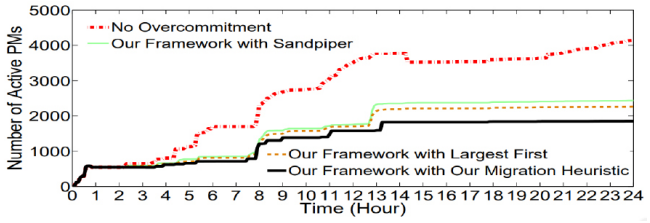

Figure 21: Number of PMs needed to host the workload.

Figure 22 shows the total energy savings when the Google cluster adapts their integrated framework (the proposed prediction approach and the proposed migration heuristic) compared to no overcommitment.

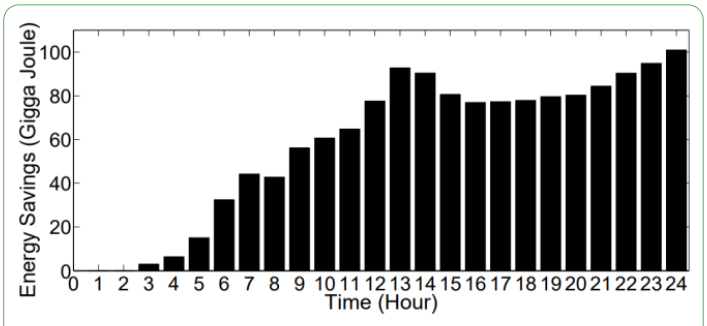

Figure 22: Energy savings our resource allocation framework achieves when compared to allocation without overcommitment.

It is clear from **Figure 22** that although their framework incurs migration energy overheads (due to both VM moving and PM switching energy overheads) that would not otherwise be present when no overcommitment is applied, the amount of energy saved due to the reduction of the number of ON PMs is much higher than the amount of energy incurred due to migration energy, leading, at the end, to greater energy savings [9].

Conclusion

Gul B et al. observed that techniques perform better with dynamic threshold as compared to static threshold, and the dynamic threshold has a positive impact on minimizing SLA violations.

Singh A et al. developed a novel VectorDot scheme to address the complexity introduced by the data center topology and the multidimensional nature of the loads on resources. Their evaluations on a range of synthetic and real data center testbeds demonstrate the validity of their system and the ability of VectorDot to effectively address the overloads on servers, switches, and storage nodes.

Gupta S and Babu MR compared the performance of a GPU with single-core and multi-core CPU (2 cores) for a basic NLP application (lexical analysis and shallow parsing). Their results show that a multi-core CPU has better performance than the single-core CPU but a GPU system has clearly overtaken them with much better performance.

Dabbagh M et al. propose an integrated energy-efficient, prediction-based VM placement and migration framework for cloud resource allocation with overcommitment. They show that their proposed framework reduces the number of PMs needed to be ON and decreases migration overheads, thereby making significant energy savings. All of their findings are supported by evaluations conducted on real traces from a Google cluster.

References

1. Makridis E, Deliparaschos K, Kalyvianaki E, Zolotas A, Charalambous T (2020) Robust dynamic CPU resource provisioning in virtualized servers. IEEE Trans Serv Comput: 1-1.
2. Singh A, Korupolu M, Mohapatra D (2008) Server-storage virtualization: Integration and load balancing in data centers. SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008.
3. Singh S, Goyal M (2015) Review paper on resource utilization and performance optimization of CPU using virtualization. Int J Adv Res Comput Commun Eng 4: 383-385.
4. Streif, RJ (2019) Containerization an alternative to virtualization in embedded systems. www.ibeeto.com, 2019.
5. Gangadhar PVSS, Venkateswara Rao M, Hota A K, Venkateswara Rao V (2017) Distributed memory and CPU management in cloud computing environment. Int J Appl Eng Res 12: 15972-15978.
6. Nema P, Choudhary S, Nema T (2015) VM consolidation technique for green cloud computing. Int J Comput Sci Inf Technol 6: 4620-4624.
7. Gul B, Khan IA, Mustafa S, Khalid O, Hussain SS, et al. (2020) CPU and RAM energy-based sla-aware workload consolidation techniques for clouds. IEEE Access 8: 62990-63003.
8. Gupta S, Babu MR (2011) Generating performance analysis of GPU compared to single-core and multi-core cpu for natural language applications. Int J Adv Comput Sci Appl 2: 50-53.

9. Dabbagh M, Hamdaoui B, Guizani M, Rayes A (2015). Efficient datacenter resource utilization through cloud resource overcommitment. 2015 IEEE Conference on Comput Commun Workshops (INFOCOM WKSHPS).

10. Jin K, Miller EL (2009) The effectiveness of deduplication on virtual machine disk images. Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference: 1-12.

11. Alla YH (2015) Time dependent virtual machine consolidation with sla (service level agreement) consideration. Master's Thesis Spring 2015.

12. Kukade PP, Kale G (2013) Auto-scaling of micro-services using containerization. Int J Sci Res 4: 1960-1963.

13. Rodriguez MA, Buyya R (2018) Containers orchestration with cost-efficient autoscaling in cloud computing environments. arXiv.org: 1-22.

14. Vayghan LA, Saied MA, Toeroe M, Khendek F (2019) Kubernetes as an availability manager for microservice applications. J Netw Comput Appl: 1-10.

15. Jimenez LL, Simon MG, Schelen O, Kristiansson J, Synnes K, et al.(2015) CoMA: Resource monitoring of Docker containers. Proceedings of the 5th International Conference on Cloud Computing and Services Science (CLOSER 2015), SCITEPRESS Digital Library 1: 145-154.

16. Shanmugam AS (2017) Docker Container Reactive Scalability and Prediction of CPU Utilization Based on Proactive Modelling. Sematic Scholar: 1-24.

17. IP Pathways (2012) Storage efficiency and the rapidly growing data center footprint. ippathways.com, 2012.