Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.29 – 34

RESEARCH ARTICLE

A Cloud Based Approach for Dynamic Resource Allocation using Virtual Machines

B. Sowmya[1], S. Vijaya Laxmi[2], Mohammed Ali Shaik[3]

[1]Pursuing M.Tech in CSE & JNTU Hyderabad, India

[2,3]Associate Professor, Department of CSE ARTI, Warangal, Telangana, India

[1] bsowmyahere@gmail.com, [2] vijaya.siru@gmail.com, [3] niharali@gmail.com

Abstract-- Cloud computing technology mainly concentrates on visualization that allows a business customer to scale up and down their resource usage based on customer needs which are never ending and the projected advantages of cloud model are those come from resource multiplexing which is based on virtualization technology. In this paper we propose a system that uses a virtualization technology for allocating data center resources dynamically which are based on application requirements and support green computing by optimizing the number of servers running or working based on the concept of skewness to measure the unevenness in the multi-dimensional resource utilization of a server by reducing skewness as much as possible by combining different types of workloads in a optimized manner which improves the overall utilization of server resources. We developed a set of heuristics that prevent overload in the system effectively while saving the energy that is being used. We implement this paper using PHP5.0 and weka 6.0 tools to simulate and shown experiment results that demonstrate our algorithms proposed performance results.

Keywords— Computing, Skewness, Virtualization, Green Computing

I. INTRODUCTION

Separation and resource control are two crucial requirements when co-locate the various types of workloads in a cloud computing environment. Separation here refers to the requirement that the execution of one workload cannot affect the execution of a different workload on the same system either in a single transaction or different transactions where as resource control refers to the ability to confine a workload to a specific set of resources in the cloud setting where as the performance separation is desirable as it is often secondary to functional and security separation wherein one workload cannot learn anything or affect the correctness of another workload or transactions. Memory capacity and processing capacity or load are two common attributes in resource control through which a workload is confined to consume no more than a certain amount of memory capacity and execute on no more than a certain number of cores or chips.

Separation and resource control have traditionally been achieved through the use of virtual machines (VM) which executes each workload inside its own virtual machine and by imposing resource constraints on the VMs that is both in terms of memory capacity and resource control are achieved. Many studies have been emerged to show how VM execution compares to native execution [1] and such studies are considered to be motivating

factor and also improvise the quality of VM technology [2]. Virtual machines are used extensively in cloud computing in present day world where the concept of infrastructure as a service (IaaS) which acts as synonym with VMs. For example, Amazon EMC^2 that makes VMs available to customers and also runs its services such as mobile aaps, databases inside VMs and many more PaaS and SaaS providers are built on IaaS that implies to run all their workloads in VMs virtually due to which the VM performance is a crucial component of overall cloud performance.

When a supervisor adds an overhead to the performance at a higher layer then VM cannot remove it hence such overhead becomes a persistent burden on cloud performance where as in a container-based virtualization technique presents an interesting alternative to virtual machines in the cloud [47]. Although the concepts underlying containers such as namespaces are very mature [36], only recently have containers been adopted and standardized in mainstream operating systems, leading to a renaissance in the use of containers to provide isolation and resource control being implemented in linux which is a preferred operating system for the cloud due to its zero price, large ecosystem, good hardware support, good performance, and reliability. The kernel namespaces feature needed to implement containers in Linux has only become mature in the last few years since it was first discussed [1]. Within the last year, Docker [3] has emerged as a standard runtime or a image format and a build system for Linux containers.

This paper aims to provide solution to two different ways of achieving isolation and control containers and virtual machines that compares the performance of a set of workloads in both the environments to that of natively executing the workload on hardware by setting benchmarks that stress different aspects such as processing time or memory bandwidth or memory latency or network bandwidth and I/O bandwidth by exploring the performance of two real time applications such as Eredis and MySQL on the different environments by providing following contributions:

- An up-to-date comparison is provided that of native container in a virtual machine environment using 2014-era hardware and software across a cross-division of interesting benchmarks and workloads that are relevant to the cloud environment.
- A number of non-obvious practical issues that affect virtualization performance are discovered in this process.
- The primary performance impact of current virtualization options for HPC and server workloads are identified.
- Containers are identified to be viable even at the scale of an entire server with minimal performance impact factor.

II. IMPLEMENTATION

The architecture of the system is presented in Fig 1. Where each physical machine runs on Xen hypervisor (VMM) which supports a privileged registered domain 0 and one or more domain U where each VM in domain U encapsulates one or more applications such as Web servers or remote desktop or DNS, etc. We assume that all physical machines share a common backend storage which multiplexes VMM into Physical machines.

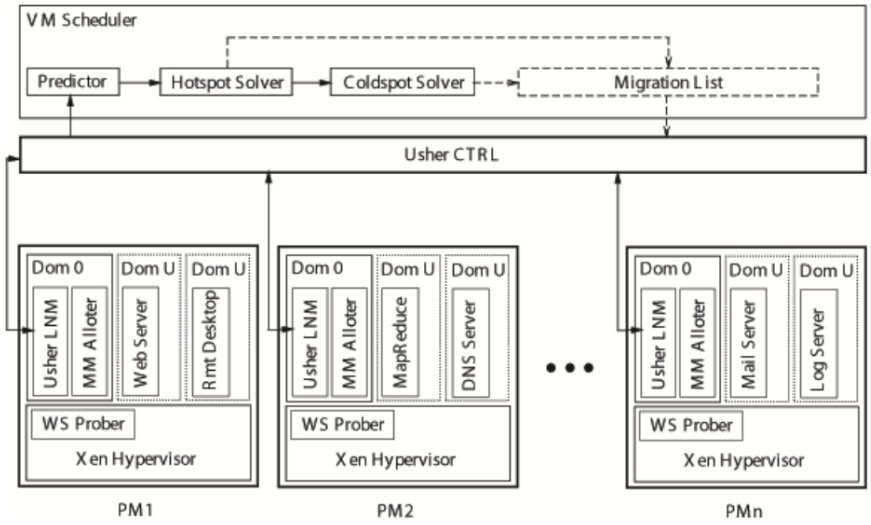

Fig. 1Architecture design.

The unix operating system traditionally does not strongly implement the principle of least privilege that is every program and every user of the system should operate using the least set of privileges necessary to complete the job and the least common mechanism principle that is every shared mechanism represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security the main objects in Unix are considered to be the file-system, processes and the network stack are globally visible to all users.

A problem caused by Unix's shared global file-system is the lack of configuration isolation where multiple applications can conflict its requirements for system-wide configuration settings where shared library dependencies can be especially problematic since modern applications use many libraries and often different applications require different versions of the same library where many applications on one operating system are installed the cost of system administration can exceed the cost of the software even.

These issues in many server operating systems lead an administrator or a developer to simplify deploy by installing each application on an separate copy of the OS either on a dedicated server or in a virtual machine which sometimes even reverses the status quo compared to a shared server on a cloud computing environment such that any code or data or configuration sharing between applications must be explicitly configured on the server.

When a customer want to get the performance one the cloud for which he or she is paying for the enterprise consolidation scenarios of the infrastructure and workload are owned by the same company or person in IaaS and PaaS there is a stronger relationship between the provider and the customer which makes it difficult to resolve performance anomalies where vitalizing the system needs to enforce resource separation to be suitable for cloud infrastructure usage.

Kernel Virtual Machine (KVM) is a feature of Linux operating system that allows Linux to act as a type 1 hypervisor by running an unmodified guest operating system (OS) inside a Linux process which uses hardware virtualization features to reduce complexity and overhead by eliminating the need for complex ring compression schemes that were pioneered by earlier hypervisors like Xen and VMware applications which are implemented using the combination of hardware acceleration and paravirtual I/O that is designed to reduce virtualization overhead to very low levels that supports live migration by allowing physical servers or even whole data centers to be evacuated for maintenance without disrupting the guest OS on which the tools get executed.

We propose that every VM is a process on which Linux resource management facilities like scheduling and cgroups are applied which simplifies the implementation and administration process of the hypervisor but complicates resource management inside the guest OS where an OS assume that CPUs are always running and memory has relatively fixed access time. VMs also have two levels of allocation and scheduling in the hypervisor and in the guest OS which are eliminated by many cloud providers due to which the problems such as over committing the resources and pinning each virtual CPU to a physical CPU by locking all virtual RAM into real RAM.

The Linux containers is the concept built on kernel namespaces feature originally motivated by difficulties in dealing with high performance computing clusters which access by the clone() system call that allows creating separate instances of previously-global namespaces on filesystem or process identifier (PID) or network or a user and hostname namespaces where each filesystem namespace has its own root directory and mount table as similar to chroot() but more powerful. A namespaces can be used specifically in creation of an isolated container that has no visibility or access to objects outside the container and when a processes running inside the container appear to be running on a normal Linux system although they are sharing the underlying kernel with processes located in other namespaces by implementing containers nest hierarchically.

All the containers need to be secured by managing Unix permissions since a container cannot access what it cannot see and thus the potential for accidentally permissions are reduced while using user namespaces by a root user inside the container which is not treated to be a root outside the container by adding additional security where the primary type of security vulnerability in containers is a system call that are not namespace-aware and thus can introduce accidental leakage between containers since the Linux system call API is huge and the process of auditing every system call for namespace-related bugs is still ongoing and such bugs can be mitigated using system call seccomp().

III. EXECUTION AND RESULTS

We introduce the concept of skewness to enumerate the unevenness in the employment of multiple resources on a server where n be the number of resources used and r_i be the utilization of the i-th resource hence we define the resource skewness of a server p as:

$$\text{Skewness(p)} = \sum \sqrt{(\frac{r_i}{r} - 1)^2}$$

In the above equation r is the average utilization of all resources for server p since not all types of resources are performance critical and hence we only need to consider major resources in the above calculation by

minimizing the skewness that can be achieved by combining different types of workloads and improve the overall utilization of server resources.

For isolating and understand the overhead which is introduced by virtual machines such as KVM and containers relative to non-virtualized and the fact is Linux can host both VMs and containers that create an opportunity for an apples-to-apples comparison between the two technologies with fewer confounding variables than many previous comparisons but we do not evaluate the case of containers running inside VMs or VMs running inside containers because we consider such double virtualization to be redundant to measure overhead we have configured our benchmarks to saturate the resources of the system under test.

VMs were configured with 32 bit vCPUs and adequate RAM to hold the benchmark's working set such as micro-benchmarks to measure CPU memory or network usage and storage overhead. We also tend to measure two real server applications such as Eredis and MySQL where these tests were performed on an Intel Xeon System with two 2.4 GHz processors and 256 GB of RAM and the processors/sockets are connected by QPI links making this a non uniform memory access (NUMA) system which is same as popular cloud providers. We also used Ubuntu 13 (Saucy) 64-bit with Linux kernel 3.11 for checking consistency of containers used an Ubuntu 13 base image.

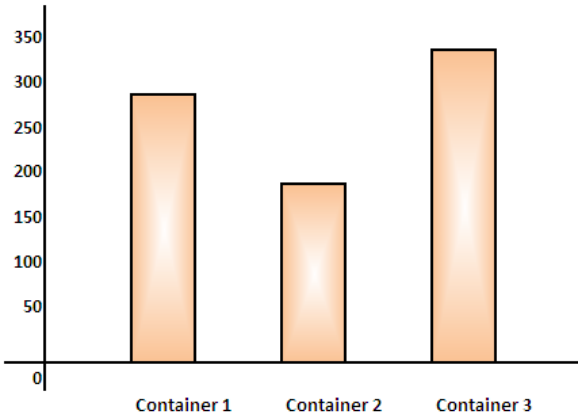

Fig. 2 performance on two sockets (16 cores) where each data point is the arithmetic mean obtained from twenty runs with error bars indicate the standard deviation obtained over all the runs.

A synthetic benchmark by name STREAM is used by us which evaluates the program that measures reliable memory bandwidth when performing simple operations on vectors to measure the memory bandwidth of the system with the working set engineered to be significantly larger than the cache memory of a system to measure the system with a much lesser extent and the cost of handling TLB misses where the memory access pattern is regular and the hardware pre-fetches typically latch on to the access pattern and prefetch data before it is needed.

Name	**Kernel**	**Bites per Iteration**	**FLOPS per Iteration**	**Hops Per iteration**
COPY	a[j]=b[j];	32	0	0
SCALE	a[j]=q * b[j];	32	1	0
ADD	a[j]= b[j] + c[j];	64	2	1
SUB	a[j]= b[j] - c[j];	64	2	1
TRIAD	a[j]= b[j] + q * c[j];	64	2	2

Table 1. Stream Components

The charts generated based on the runs are:

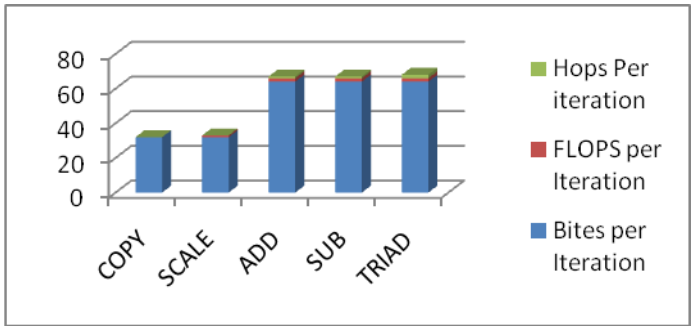

Fig. 3 Stream performance in GBs on one socket with eight cores where each data point is the arithmetic mean obtained from twenty runs and the error bars indicate the standard deviation obtained over all runs.

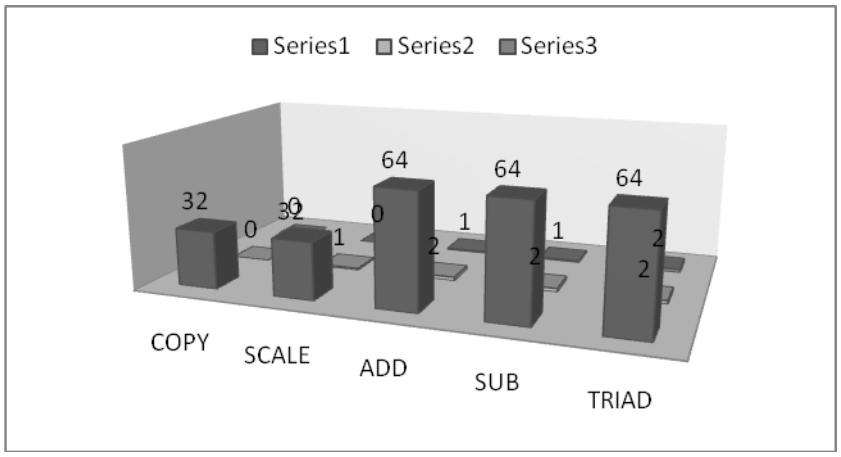

Fig. 4 Stream performance in GBs on both sockets with all 16 cores where each data point is the arithmetic mean obtained from twenty runs where error bars indicate the standard deviation obtained over all runs.

The green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold value set with a pre-defined value based on which we sort the list of cold spots in the system based on the ascending order of their memory size for migrating away all its VMs before we can shut down an under-utilized server where we define the memory size of a cold spot as the aggregate memory size of all VMs running on 16 or 32 processors running on the core by which the cost of a VM live migration is determined mostly by its memory footprint.

We chose to evaluate MySQL because it is a popular relational database that is widely used in the cloud computing environment and it stresses memory and file system subsystems we executed the SysBench, OLAP benchmark against a single instance of MySQL 5.5 where the the backend stores data of 3GB cache with capacity to perform all reads during the benchmark runs that comprises of a database preloaded with 2 million records and executes a fixed set of read/write transactions choosing between four SELECT queries and with two UPDATE queries and a single DELETE query and an INSERT query. The resulting measurements provided by SysBench are statistics of transaction latency and throughput in transactions per seconds or per CPU cycle where the number of clients was varied until saturation and ten runs were used to produce each data point.

IV. CONCLUSION AND FUTURE WORK

We have proposed and presented the design, implementation, and evaluation of a resource management system for cloud computing services where in our system multiplexes virtual to physical resources adaptively based on the changing demand we use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized and we achieved both overload avoidance and green computing for systems with multi-resource constraints.

We even found KVM performance has improved considerably for the challenging workloads with line-rate of 10 Gbps network is possible using only a single core using 2014-era hardware and software where KVM still

adds some overhead to every I/O operation which ranges from significant when performing small I/Os to negligible when it is amortized over large I/Os hence KVM is less suitable for workloads that are latency-sensitive or have high I/O rates.

REFERENCES

[1] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. Quantifying the performance isolation properties of virtualization systems. In Proceedings of the 2007 Workshop on Experimental Computer Science, ExpCS '07, 2007.

[2] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux virtual machine monitor. In Proceedings of the Linux Symposium, volume 1, pages 225–230, Ottawa, Ontario, Canada, June 2007.

[3] S. Soltesz, H. P ¨ otzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, EuroSys '07, pages 275–287,2007.

[4] Implementing IBM FlashSystem 840.

[5] Sysbench benchmark. https://launchpad.net/sysbench.

