Resource usage monitoring for KVM based virtual machines

Ankit Anand, Mohit Dhingra, J. Lakshmi, S. K. Nandy
CAD Lab, Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
ankit@ssl.serc.iisc.in, mohit@cadl.iisc.ernet.in, {jlakshmi, nandy}@serc.iisc.in

***Abstract*—Realization of cloud computing has been possible due to availability of virtualization technologies on commodity platforms. Measuring resource usage on the virtualized servers is difficult because of the fact that the performance counters used for resource accounting are not virtualized. Hence, many of the prevalent virtualization technologies like Xen, VMware, KVM etc., use host specific CPU usage monitoring, which is coarse grained. In this paper, we present a performance monitoring tool for KVM based virtualized machines, which measures the CPU overhead incurred by the hypervisor on behalf of the virtual machine along-with the CPU usage of virtual machine itself. This fine-grained resource usage information, provided by the above tool, can be used for diverse situations like resource provisioning to support performance associated QoS requirements, identification of bottlenecks during VM placements, resource profiling of applications in cloud environments, etc. We demonstrate a use case of this tool by measuring the performance of web-servers hosted on a KVM based virtualized server.**

Keywords*—*Monitoring, Performance Analysis, Virtual machine monitors, System performance, Clouds.

I. Introduction

Most of the enterprises, today, have applications that have resource centric usage like I/O specific or CPU specific. Above this, the application workload also exhibits dynamic behavior based on time of the day or geographic locality. In such situations, to improve resource utilization, many enterprises are shifting towards cloud computing solutions where elastic resources can be availed on a pay-by-use mode. Virtualization is a key enabler in these cloud based solutions where multiple applications are co-hosted on a single machine. On such virtualized servers, physical hardware is shared among multiple virtual machines by a layer called hypervisor or virtual machine monitor(VMM). Applications in turn, are hosted in the virtual machines and access the underlying shared hardware through hypervisor.

Virtualization helps provide software isolation to applications in such shared environments as well as ensures better utilization of server resources. On cloud systems, I/O intensive applications are good candidates for virtualization because they have sufficient spare CPU cycles which can potentially be used by some other co-hosted application.

Many virtual machine monitors have emerged in such a scenario, varying from VMware ESX to Xen paravirtualized hypervisor. In recent years, to make development of virtual machine monitors easy, hardware vendors like AMD and Intel have added virtualization extensions to x86 processors which were initially difficult to virtualize and were not in tune with Popek and Goldberg virtualization requirements [1].

The Kernel based Virtual Machine (KVM) is a relatively new VMM which utilizes these hardware extensions and has found its way in Linux kernel. It is a full virtualization solution, which requires no changes in guest operating system[2].

While virtualization provides a simple mechanism to share resources by isolating the application's software environment, most of the applications incur some kind of virtualization overhead. This overhead varies depending upon the type of application, type of virtualization and virtual machine monitor used. Particularly, for I/O applications, the overhead of CPU used by VMM on behalf of VMs is considerable and affects performance characteristics of applications [3][4].

To account for such overhead, it is necessary to monitor resources used by the hypervisor on behalf of the VM. Resource monitoring in virtualized scenario is not straight-forward since the hardware performance counters are not virtualized and hence cannot be visible to the GuestOS in the VM[5].However, these performance counters can be accessed from inside the hypervisor where details of VM specific usage is not visible. What is required is a correlation of the usage within the VM to that of the hypervisor to support this usage. Currently, there are many tools available in Xen, KVM, VMware etc. which gives host specific resource usage. This is coarse grained since it depicts which host is saturating or has idle resources but not how a VM is using those resources. To overcome this deficiency, the xenmon [6] tool was developed for the Xen hypervisor. In Xen [4], the hypervisor usage on behalf of all virtual machines is monitored based on the VM-labeling of the activity from within the Dom 0, the privileged VM in Xen. But in the case of KVM, the same is not possible since hypervisor is a part of VM process and not a separate virtual domain in itself. The key to monitoring VM-specific usage in KVM is by tracing these VM-processes and accounting for their resource usage. In this paper, we present a simple resource monitoring tool for measuring CPU resource usage of hypervisor on behalf of each VM and CPU usage of VMs separately in the case of KVM process.

The rest of the paper is organized as follows. Section II describes the KVM architecture with a focus on monitoring. Section III describes the existing monitoring tools available and how the same can be leveraged to achieve the desired objective. Section IV describes the design model of tool and section V presents a use case of the tool designed. Finally, Section VI concludes by discussing the potential applications of information so obtained and future work.

II. KVM Architecture

KVM is a more recent hypervisor which embeds virtualization capabilities in Linux kernel using x86 hardware virtualization extensions[2][7]. It is a full virtualization solution, where guests are run unmodified in VMs. It consists of two modules, namely, kvm.ko and an architecture dependent kvm-amd.ko or kvm-intel.ko module. Under KVM, each VM is spawned as a regular linux process named KVM and scheduled by the default linux scheduler. For KVM the hardware has to support three processor modes, namely user, kernel and guest mode. The guest mode is added to support hardware assisted virtualization. The virtual machine executes in this guest mode which in turn has user and kernel mode in itself [8][9].

For using shared I/O hardware, these VMs interact with Qemu emulator in host user space which provides emulated I/O devices for virtual machines. For instance, in the case of network related applications, Qemu provides emulated Network Interface Card (NIC) to VMs and interacts with tun-tap device on the other side. The tap device is connected to physical NIC through a software bridge.

Fig 1 shows the typical KVM architecture, with reference to a network related application. A typical network packet flows through the KVM virtualized host in the following way. As depicted in picture, when a packet arrives at physical NIC, interrupts generated by NIC are handled by the physical device driver. The device driver forwards the packet to software bridge. The bridge, then pushes the packet to the tap device of the corresponding VM. The tap device is a virtual network device that sends a signal to KVM module. KVM module in turn, generates a virtual interrupt to the user space Qemu of the target VM. Qemu then copies the packet from tap device and generates the interrupt for the guest OS emulating the virtual NIC. Again, the physical device driver in the guest OS handles the packet transfer to the VM's address space.

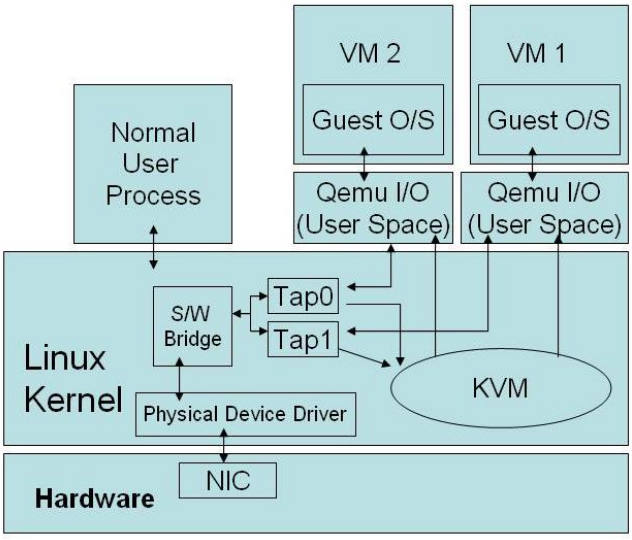

Fig. 1. KVM Architecture (Source [8])

Consequently, for a VM process in KVM virtualized server, guest mode execution (both kernel or usermode) corresponds to execution within a VM while other modules in user mode (Qemu) and kernel mode (KVM module, tun-tap module, bridge module, etc.) correspond to hypervisor execution. Among these, Qemu I/O module runs separately for each VM but is co-ordinated by a single KVM module which manages VMs by signal and virtual interrupts. Hence, it is easy to understand that the KVM module can potentially become a bottleneck when it tries to execute on behalf of many VMs.

III. Related Work and Monitoring Tools available

Many resource and performance monitoring tools are available for non-virtualized systems. The type of tool to be used depends on the granularity of information to be extracted and frequency of profiling. Since, here, we are concerned with resource monitoring in KVM based VMs, we will be discussing available Linux based monitoring tools and that too, in open source domain since KVM itself is an open source software.

Most of the commonly used tools include software profilers like OProfile, gprof, top etc.. Some of these use underlying hardware performance counters to profile the system. Such performance monitoring tools are available in almost all operating systems, some (like top) perform coarse grained monitoring while others like OProfile collect fine grained information.

In virtualized systems, the task of profiling and resource monitoring is not straight-forward because Performance Monitoring Unit (PMU) is not virtualized [10] [5]. Especially, with hardware assisted virtualization, CPU can enter in another mode called guest mode which has to be taken separately into account.

The existing tools like OProfile [11], top etc. give the total CPU usage of a particular KVM VM which includes the CPU usage of both the hypervisor as well as VM. Similarly, tools like XenMon which are specific to Xen VMM gives total CPU usage of Xen hypervisor i.e Dom 0 but don't separate it into hypervisor usage per VM. In our previous work [4], we developed a generic monitoring framework, and illustrated the same for Xen hypervisor. This monitoring framework extracts hypervisor CPU usage on behalf of each VM by counting the number of page flips between Dom 0 (hypervisor) and Dom U (virtual machine) using OProfile. The same mechanism does not work for KVM due to difference in architecture of two VMMs.

Apart from this, certain tools exist in cloud computing frameworks like OpenNebula[12] which only provides coarse grained information of percentage of Virtual CPU used by a VM. Also, dedicated monitoring systems like Ganglia [13], a performance monitoring system for high performance applications is used in clouds to extract information inside VMs. Other industry standards like Nagios [14] provides VM process level details but does not again account for the hypervisor's overheads for a VM.

One of the tool which is in close match with KVM in this context is Linux Perf [15]. Perf is a profiling tool for linux based systems and hence makes a good integration with KVM which is also embedded in Linux kernel itself. Perf internally uses hardware performance counters for profiling. Initially, certain events like instruction execution, cache misses etc. are

selected based on which profiling needs to be done. A counter is incremented whenever such an event occurs and when the counter reaches a predefined value, an interrupt is generated and the program counter value at that time is recorded, with counter being reset again. The interpretation of recorded events gives the percentage CPU used by different programs at a finer granularity of function level.

Perf, apart from profiling like OProfile, has a special command "perf kvm" which can be used to profile the guest kernel much like the host kernel. Also, it gives a clear percentage of CPU used in guest mode, host user and host kernel mode. In addition, we can get the CPU profile of guest kernel as well.

Using the architectural details of KVM and the above information provided by Perf KVM, we extract the CPU usage of hypervisor in each module on behalf of a particular VM. The next section describes system model for the same.

IV. System Model for Resource Monitoring

The basic system model is shown in Fig. 2. As described in Section II, each VM under KVM hypervisor is a Linux process. Initially, we extract the PIDs of VM linux processes named KVM.

Then, for a given monitoring period, resource profiles are generated across all CPUs by a profile recorder (like "perf kvm record") and fed into the system along with PIDs. Using these PIDs, we segregate the process wise CPU usage using the corresponding interpreter of the profile recorder used in initial phase. In our case, we have used perf kvm report for the same. The important point to note is that profiler must be enabled in virtualization mode. For example, we have to use suffix "kvm" to record profiles for guest mode separately in case of perf.

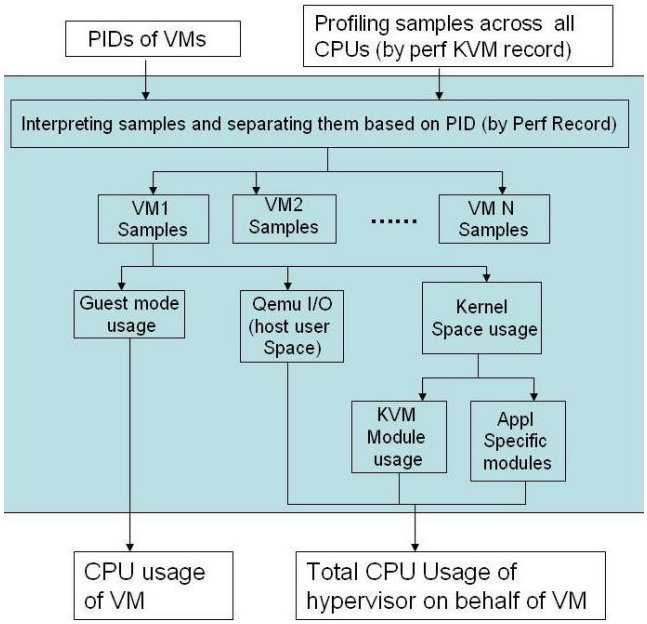

Fig. 2. System Model for Resource Monitoring

Then, resource usage profiles obtained for VMs are used to obtain CPU usage in the guest and host user and kernel modes. The guest mode usage (including both guest kernel and guest user mode) accounts for total CPU used by VM where as CPU usage of hypervisor consists of Qemu (host user mode) and multiple kernel modules like KVM core modules(kvm.ko and kvm_amd.ko/kvm_intel.ko), , TUN or TAP device module, etc. specific to application. The only issue is with regard to the resource accounting for the physical device driver and the bridge module. This usage can be distributed in proportion to the network bandwidth used by each of the VMs. Adding total CPU usage of all modules in a VM, we get total hypervisor effort on behalf of that VM.

The important point to note in case of KVM hypervisor is that even if we pin VMs to specific cores, guest mode usage(VM) is tied to that core only but hypervisor keeps on switching across multiple CPUs present on host machines and so profiling must be done across all working CPUs on the host machine so as to achieve correct hypervisor usage.

The overhead to record profiling information is just the overhead of running perf in the system which has less than 1 percent overhead on the host system which is negligible. Subsequent resource accounting that is done by shell scripts is also a very light operation which can be neglected for considerable monitoring time. So, our tool as a whole does not cause any specific overhead which can change the profiling information.

The above system design is not specific to perf but suited for any general profiler provided it supports granularity at module level and takes into account guest mode usage separately. The components of perf can be replaced with the corresponding components in such a case. The following section illustrates a use case of tool that is described above.

V. Use Case

In this section, we present a use case of tool described in previous section in a KVM virtualized host. Here, we aim to study the behavior of web-servers hosted inside virtual machines residing on a cloud system. We choose this application because web servers being I/O intensive workloads are one of the most commonly hosted applications in clouds and present a good case scenario in terms of different resource usage.

For experimentation, httperf [16] is used as a tool for generating representative workload requests for the web servers. It runs on client machines and generates specified number of requests for web-servers in the form of requests per second. The performance characteristics of servers are measured in the form of statistics associated with average response time to a request, throughput as number of replies generated against the given workload, no. of errors etc.. By varying the generated workloads, we analyze the server physical resource usage, response time and throughput. Response time gives a performance metric of the application. Infact, it is one of the most important metric governing performance Service Level Agreements(SLAs). So, we intend to co-relate fine grained physical resource usage information with performance characteristics of the application.

A. Experimental Setup

The table I shows the characteristics of physical machine and virtual machine in our set-up. OpenNebula Cloud environ-

TABLE I. MACHINE SPECIFICATIONS

Hw-Sw	Physical Machine	Virtual Machine
Processor	AMD 2.4 Ghz 12 cores	AMD 2.4GHz 1 core
Memory	16GB	3GB
OS	OpenSuse 12.1	Opensuse 11.4
Network Bandwidth	1Gbps	Best Effort

ment is used, with KVM full virtualization as hypervisor, to create virtual machines [12]. Also, for the experiment, we have pinned each of VMs to a single core. But in this case,since hypervisor is a part of VM process, we can't pin the hypervisor separately to a particular core. Also, CPU frequency scaling is switched off to capture correct hypervisor usage across all cores.

We ran the test by varying request rates and number of virtual machines on the host. In the next subsection, we analyze the results of performance and resource monitoring of application.

B. Results

We conduct the experiment for non-virtualized case, virtualized cases with 1 VM, 2 VMs and 3 VMs and gather information on different application characteristics like response time and throughput. And, for these metrics we also measure the ensuing resource usage by the VM and the hypervisor on behalf of the VM.

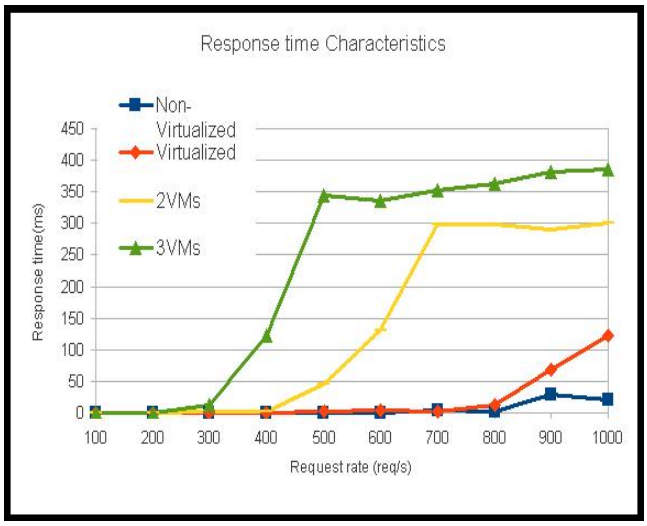

Fig. 3. Response time characteristics

Figure 3 shows the response time for different cases when we vary request rate from 100 req/s to 1000 req/s. There is not much difference in case of non-virtualized to virtualized case but response time begins to increase sharply in case of 2 VMs at request rates of around 400 and even early in case of 3 VMs. Similar trends were observed in throughput characteristics (figure 4). Here, we observe that although each virtual machine was provided with a separate CPU, the performance metric drops to 40 percent of single VM case. This must happen due to some resource exhaustion in the system.

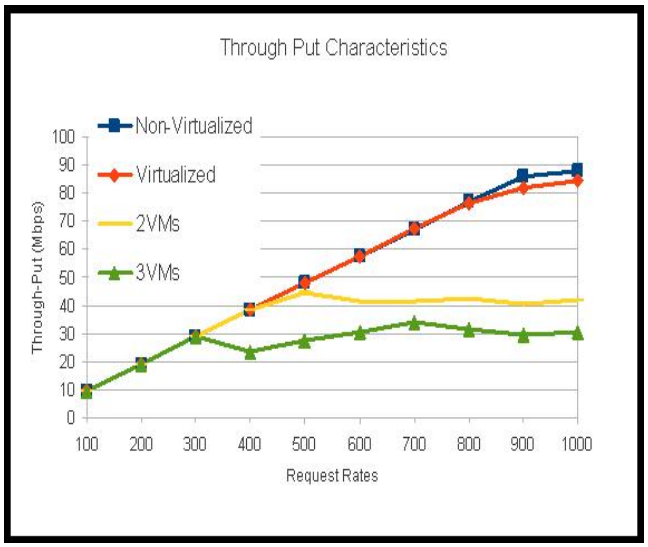

Fig. 4. Throughput characteristics

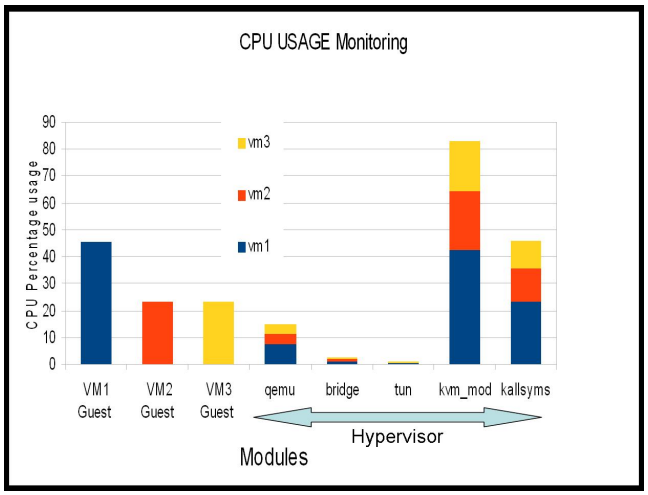

Fig. 5. CPU Usage -3 VMs

To explain the performance degradation, we analyzed usage of different resources in the system. To analyze CPU usage, we ran the KVM-Perf tool explained in previous section and obtained CPU percentage used by different modules of hypervisor on behalf of all VMs. Figure 5 shows the output of tool for 3 VMs case. Here, we are showing the guest space usage for each VM and hypervisor usage on behalf of each VM in different modules. The hypervisor usage per module on behalf of each VM is stacked together for all VMs and is shown with the corresponding color of guest space usage. Here we observe, the total hypervisor usage exceeds more than 100 percent where as VM CPU usage is less than 50-60 percent. Analyzing module by module, we see that the kvm module (kvm.ko and kvm-amd.ko), which executes in a serial fashion, consumes more than 80 percent CPU at around 300 req/s in 3 VM case. Also, at corresponding rates, response time starts to increase sharply. We infer that, hypervisor CPU usage is a different resource from CPU allocated to VMs and it can prove to be a critical resource in certain cases, thereby affecting the performance drastically.

Also, this information of CPU usage of hypervisor on per VM basis as provided by tool helps us to analyze an overloaded VM which may be causing bottleneck in the whole system. The same will help to take better resource provisioning decisions for associated Quality-of-Service (QoS) requirements.

VI. Conclusion and Future Work

In this work, we have come up with a resource usage monitoring tool for KVM, which can provide hypervisor CPU usage per VM. The above fine grained monitoring information can be put to use in diverse situations varying from resource provisioning to support performance associated QoS requirements, identifying bottlenecks in the system, as illustrated above, and performance profiling of VMs etc.. As per authors' knowledge, no existing cloud computing frameworks like OpenNebula, Eucalyptus etc. extract such fine grained resource monitoring information which proves to be critical and valuable in making certain decisions. In future, we intend to integrate this tool with cloud computing frameworks like OpenNebula, Eucalyptus etc. so that, information extracted from monitoring unit could be used for better resource provisioning decisions and improving the efficiency of system as a whole.

References

[1] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361073

[2] (2012) Main page-kvm. [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[3] J. Lakshmi, "System virtualization in the multi-core era, a qos perspective," Dissertation, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, 2010.

[4] M. Dhingra, J. Lakshmi, and S. Nandy, "Resource usage monitoring in clouds," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, sept. 2012, pp. 184 –191.

[5] J. Du, N. Sehrawat, and W. Zwaenepoel, "Performance profiling of virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '11. New York, NY, USA: ACM, 2011, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/1952682.1952686

[6] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," Tech. Rep., 2005.

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

[8] S. Zeng and Q. Hao, "Network i/o path analysis in the kernel-based virtual machine environment through tracing," in *Information Science and Engineering (ICISE), 2009 1st International Conference on*, dec. 2009, pp. 2658 –2661.

[9] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan, "Performance analysis towards a kvm-based embedded real-time virtualization architecture," in *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, 30 2010-dec. 2 2010, pp. 421 –426.

[10] A. Khandual, "Performance monitoring in linux kvm cloud environment," in *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, oct. 2012, pp. 1 –6.

[11] (2012) Oprofile - a system wide profiler for linux. [Online]. Available: http://oprofile.sourceforge.net/news/

[12] (2012) Opennebula- the open source solution for data center virtualization. [Online]. Available: http://opennebula.org/

[13] (2012) Ganglia monitoring system. [Online]. Available: http://ganglia.sourceforge.net/

[14] (2012) Nagios. [Online]. Available: http://www.nagios.org/

[15] (2012) Main page-perf. [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[16] D. Mosberger and T. Jin, "httperf-a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998. [Online]. Available: http://doi.acm.org/10.1145/306225.306235