RT-OpenStack: CPU Resource Management for Real-Time Cloud Computing

Sisu Xi, Chong Li, Chenyang Lu, Christopher D. Gill
Cyber-Physical Systems Laboratory
Washington University in St. Louis
{xis, lu, cdgill}@cse.wustl.edu, chong.li@wustl.edu

Meng Xu, Linh T.X. Phan, Insup Lee, Oleg Sokolsky
University of Pennsylvania
{mengxu, linhphan, lee, sokolsky}@cis.upenn.edu

***Abstract*—Clouds have become appealing platforms for not only general-purpose applications, but also real-time ones. However, current clouds cannot provide real-time performance to virtual machines (VMs). We observe the demand and the advantage of co-hosting real-time (RT) VMs with non-real-time (regular) VMs in a same cloud. RT VMs can benefit from the easily deployed, elastic resource provisioning provided by the cloud, while regular VMs effectively utilize remaining resources without affecting the performance of RT VMs through proper resource management at both the cloud and the hypervisor levels. This paper presents *RT-OpenStack*, a cloud CPU resource management system for co-hosting real-time and regular VMs. RT-OpenStack entails three main contributions: (1) integration of a real-time hypervisor (RT-Xen) and a cloud management system (OpenStack) through a real-time resource interface; (2) a real-time VM scheduler to allow regular VMs to share hosts with RT VMs without interfering the real-time performance of RT VMs; and (3) a VM-to-host mapping strategy that provisions real-time performance to RT VMs while allowing effective resource sharing with regular VMs. Experimental results demonstrate that RT-OpenStack can effectively improve the real-time performance of RT VMs while allowing regular VMs to fully utilize the remaining CPU resources.**

I. Introduction

An important advantage to run real-time applications in a cloud is its abundant computing resources, which makes cloud computing an attractive choice for hosting computation-intensive real-time tasks, such as object recognition and tracking, high-definition video and audio stream processing, and feedback control loops in general. For example, a gaming console can use the computing power in the cloud to provide better image quality to the end user [1]. Prolonged latency for such applications often leads to frustrating or unacceptable experience to end users. Such applications therefore require latency guarantees.

Despite the growing demand for running real-time applications in the cloud, however, current clouds provide limited support for real-time performance. Most clouds allow users to specify only the number of Virtual CPUs (VCPUs) associated with a VM and/or their CPU shares. Furthermore, cloud management systems often oversubscribe the system to better utilize the resources. As a result, if a co-locating VMs consumes lots of resources, other VMs on that host will suffer performance degradation, which is known as the "noisy neighbor" problem.

The lack of cloud resource management for latency guarantees has led cloud providers and users to develop proprietary application-level solutions to cope with resource uncertainty. For example, Netflix, which runs its services in Amazon EC2, constantly monitors the resources used by each VM. If one VM cannot meet its performance requirement (usually due to a co-located noisy neighbor), Netflix shuts down the VM and restarts it on another host, hoping that the newly located host is less crowded [2]. Moreover, Netflix developed a tool called "chaos monkey" [3], which introduces artificial delays to simulate service degradation and then measures whether the application can respond appropriately. An alternative solution is to pay for dedicated hosts for running real-time applications, which usually results in resource under-utilization and may not be cost-effective.

This paper presents *RT-OpenStack*, a CPU resource manager for co-hosting real-time and non real-time (regular) VMs, RT-OpenStack makes three main contributions: (1) integration of a real-time hypervisor (RT-Xen) and a cloud management system through real-time resource interfaces; (2) a real-time VM scheduler allowing effective resource sharing between regular and real-time VMs while maintaining real-time performance of real-time VMs; and (3) a VM-to-host mapping strategy that provisions real-time performance to real-time VMs while allowing effective resource sharing by regular VMs. We have implemented and evaluated RT-OpenStack by extending the OpenStack cloud manager and the Xen hypervisor, in principle our approaches may be extended to other cloud management systems and hypervisors.

The rest of the paper is structured as follows: In Section II we introduce background about Xen and OpenStack, and their key limitations in supporting real-time applications. After discussing related work in Section III, we present the design and implementation of RT-OpenStack in Section IV and experimental evaluation in Section V. We conclude the whole paper in Section VI.

II. Background

We first introduce Xen and our previous work on RT-Xen. We then review the OpenStack cloud management system and its scheduling components. We identify their limitations in supporting real-time applications alongside general purpose applications which motivate the design of RT-OpenStack.

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.33

A. Xen Virtual Machine Monitor

Xen [4] is a type-1 (bare metal) open-source virtual machine monitor (VMM)[1] used in commercial clouds, such as Amazon EC2 and RackSpace. It sits between the hardware and operating systems. From the scheduling perspective, each virtual machine (called a domain in Xen) contains multiple VCPUs that are scheduled by a VMM scheduler on a host with multiple physical CPUs (PCPUs). At boot time, Xen creates a privileged domain called domain 0, which is responsible for managing the other guest domains. By default, Xen uses a credit scheduler based on a proportional-share scheduling policy: each VM is associated with a *weight*, which represents the share of CPU resource it will receive relative to other VMs. The system administrator can also specify a *cap* per VM, which is the maximum CPU resource that can be allocated to each VM. As the credit scheduler does not consider timing constraints of the applications, it is not suitable for real-time applications [5, 6]. To support real-time applications, the recently released Xen 4.5 include a real-time scheduler called RTDS [6] developed in the RT-Xen project described in the next subsection.

B. RT-Xen

We have designed and implemented RT-Xen [5, 7, 6], a real-time scheduling framework for Xen. In RT-Xen, each VCPU specifies its demand for CPU resources as a resource interface, which includes three parameters: *budget* (the amount of time a VCPU is allowed to run per period); *period*; and *cpumask*, the set of the PCPUs on which the VCPU is allowed to run. The current multi-core scheduler in RT-Xen [6] supports a rich set of real-time scheduling policies including earliest deadline first (EDF) and rate monotonic (RM) priority schemes, global and partitioned scheduling policies, and different budget management schemes such as deferrable and periodic servers. Among all the combinations of real-time scheduling policies, global EDF (gEDF) with deferrable server delivered the best real-time performance in our experiments [6]. From Xen 4.5, the gEDF with deferrable server is included in the Xen release as the RTDS scheduler [8].

However, RT-Xen has two drawbacks in supporting RT VMs in a cloud. First, RT-Xen employs compositional scheduling analysis (CSA) [9] to compute the resource interfaces of VCPUs needed to guarantee the real-time performance of applications running in the VMs. CSA represents the resource requirements of each VM based on the multiprocessor periodic resource model, $\mu = \langle \Pi, \Theta, m' \rangle$, in which in every period of Π time units, the resource allocation provides a total of Θ execution time units, with a maximum level of parallelism m'. The CSA interface naturally maps to a VM with m' VCPUs with a period of Π ms and a total budget of Θ ms. While CSA provides real-time guarantees on RT-Xen, the resource interfaces computed based on the CSA are often conservative due to the pessimism of its schedulability analysis. As a result, provisioning CPU resources based on the resource interfaces may lead to significant CPU under-utilization [6]. Second, there is no distinction between RT and regular VMs. Both RT and regular VMs are scheduled using the same type of resource interface, and the regular VMs must be incorporated into the underlying compositional schedulability analysis even though they do not require latency guarantees. Thus, if we directly apply the current RT-Xen in a cloud, the host may be under-utilized.

[1] We use the terms hypervisor and VMM interchangeably.

C. OpenStack

OpenStack [10] is a popular cloud management system. It adopts a centralized architecture consisting of interrelated modules that control pools of CPU, memory, networking, and storage resource of many machines. When integrated with Xen, a special agent domain is created on each host to support these resource management functions in co-ordination with domain 0 of the host.

We now review three aspects of OpenStack that are critical for managing the performance of VMs: (1) the resource interface that specifies the resource requirement of a VM; (2) the admission control scheme for each host to avoid overload; and (3) the VM allocation scheme that maps VMs to hosts:

Resource Interface: The resource interface in OpenStack is represented by a pre-set type (called a "flavor"). The cloud manager can configure the number of VCPUs, memory size, and disk size. A user cannot specify the resource demand of each VCPU, which may be necessary to provide real-time performance guarantees.

Admission Control: Admission control in OpenStack is referred to as "filtering". OpenStack provides a framework where users can plug-in different filters. Many filters are provided for checking sufficient memory, storage, as well as for VM image compatibility. Two of the filters are related to the CPU resources: (1) the core filter, which uses a VCPU-to-PCPU ratio to limit the maximum number of VCPUs per host (by default, this ratio is set to 16:1); (2) the max VM filter, which limits the maximum number of VMs per host (by default, this value is set to 50). Clearly, these filters cannot provide real-time performance guarantees to real-time VMs because it ignores the different resource demands and latency requirements among VMs.

VM Allocation: After the filtering process, OpenStack needs to select a host to place the VM. This is referred to as "weighing". By default, OpenStack uses a worst-fit algorithm based on the amount of free memory on each host. This policy again does not take into account the CPU resource demands of different VMs.

While OpenStack is widely used in general purpose cloud, it cannot support real-time VMs demanding latency guarantees. First, the resource interface is inadequate. A user can configure only the number of VCPUs, and cannot specify the resource demand needed to achieve real-time performance guarantees. Moreover, the VM allocation heuristics similarly ignores the CPU resources needed to meet the real-time performance requirements.

III. Related Work

This paper focuses on co-hosting RT VMs with regular VMs, both on a single host (RT-Xen 2.1) and in a public cloud (RT-OpenStack). We now discuss related work in both areas.

There is a rich body of theoretical results on hierarchical real-time scheduling [11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

These approaches cover different aspects of the problem, and differ in the scheduling policies and resource interfaces used in a multi-core environment. However, none of them considers the problem of co-hosting RT VMs with regular VMs.

Earlier works on real-time hypervisor scheduling [21, 22, 23] employ heuristics to enhance the default schedulers in Xen. RT-Xen provides a new real-time scheduling framework to plug-in different real-time schedulers based on compositional scheduling analysis. One of the multi-core RT-Xen schedulers called RTDS [6] has recently been included in Xen 4.5 as an experimental feature. While these earlier efforts on RT-Xen aims to support real-time VMs only, the RT-Xen 2.1 scheduler introduced in this paper represents a new contribution by co-scheduling RT and regular VMs while maintaining the real-time guarantees to RT VMs. While the implementation of RT-Xen is specific to the Xen platform, the approach can be easily applied to other virtualization technologies like KVM [24, 25] and micro-kernels [26, 27, 28].

How to provide real-time performance with regards to other resources like cache, memory, and network I/Os have also been studied theoretically and/or practically. For example, [19] presented a cache-aware compositional analysis to ensure timing guarantees of tasks running on a multi-core virtualized platform; and [29] provided a modification to domain-0 to prioritizing network traffic for different VMs. RT-OpenStack and RT-Xen can be extended to consider these resources by using existing solutions like dominant resources [30], which we leave to future work.

VMWare vCenter [31] maintains each host's utilization between 45% and 81%, and dynamically powers up or shut down standby hosts to save energy. The vCenter also performs VM live migration to balance the load between multiple hosts. We plan to investigate VM migration as future work. For open source cloud management systems, the most related work is [32], which also migrates VM to balance the load between hosts. In contrast, RT-OpenStack addresses the VM placement problem to provide real-time guarantees to real-time VMs.

IV. Design of RT-OpenStack

As the first step toward a cloud resource management for real-time VMs, we have developed RT-OpenStack, a CPU resource manager for real-time and regular VMs sharing a common cloud computing platform. [2] The salient feature of RT-OpenStack lies in its capability to meet the real-time performance requirements of real-time VMs, while allowing regular VMs to effectively share remaining CPU resources in a cloud computing platform. Specifically, RT-OpenStack is designed based on the following principles:

- It should support a resource interface that allows a real-time VM to specify the amount and temporal granularity of CPU resource allocation needed to meet the real-time performance requirements of its applications.
- It should guarantee the CPU resources provisioned to each real-time VM according to its resource interfaces.
- It should perform real-time-aware VM-to-Host mapping to maintain the schedulability of real-time VMs without overloading the hosts.
- It should be work-conserving and allow regular VMs to effectively utilize remaining CPU resources without affecting the real-time performance of real-time VMs.

RT-OpenStack integrates real-time VM scheduling at the host level and CPU resource management at the cloud level. At the host level, we designed RT-Xen 2.1, a real-time VM scheduler in the Xen hypervisor that is specifically designed for effective resource sharing between real-time and regular VMs on a same host, while maintaining real-time performance of real-time VMs. At the cloud management level, we developed a new VM-to-host mapping strategies to meet the real-time performance requirements of real-time VMs while allowing regular VMs to utilize remaining CPU resources in the cloud computing platform.

In the following parts, we first describe the co-scheduling RT and regular VMs at a single host level, and then detail the process of the VM-to-host mapping for both RT and regular VMs. Finally, we provide the implementation details on how to pass RT VMs' information between OpenStack and Xen hypervisor.

A. Hypervisor scheduling policy

We designed RT-Xen 2.1 to co-schedule RT and regular VMs on a same host. The key difference between RT-Xen 2.1 and previous RT-Xen is that RT-Xen 2.1 can integrate regular VMs into the scheduling framework without interfering RT VMs' performance. There are three key changes. First, the resource interface are now represented by four parameters: *budget* (the amount of time a VCPU is allowed to run per period); *period*; *cpumask*, the set of the PCPUs on which the VCPU is allowed to run; and *rt*, indicating this VM is RT or regular VM. With the new *rt* flag, RT-Xen 2.1 implements a strict fix priority scheduling where the regular VM always has lower priority than RT VMs. Second, we organize the run queue as shown in Figure 1. A run queue holds all VCPUs with tasks running, and the scheduler always picks the head (as allowed by *cpumask*) to make the scheduling decision. We divide the VCPUs into two categories: with or without of *budget*. Within each category, we strictly prioritize the real-time VMs' VCPUs over the regular VMs' VCPUs. Therefore, the regular VMs do not affect the compositional schedulability analysis of the real-time VMs. At the same time, they can utilize the remaining CPU resources. Third, for RT VMs, we use the gEDF scheduling policy with a deferrable server, which had the best performance in our previous experiments [6] and was included in Xen 4.5; and for regular VMs, we use a round-robin scheduling policy between them. We can apply more sophisticated scheduling policies for regular VMs, but is deferred as future work.

B. VM-to-host scheduling policy

Another important part is the VM-to-host mapping when creating a VM. Recall that OpenStack divides this process into two phases: "filtering" and "weighing". For regular VMs, we use existing filters and weigher as the default setup; For RT VMs, we use existing filters plus a RT-Filter in the filtering

[2]The current RT-OpenStack focuses on CPU resource management which has significant impacts on VM latency. The other resources such as storage and networking can also impact latency and will be addressed in future work.

TABLE I: RT-OpenStack for real-time and regular VMs

	Resource Interface	Admission Control	VM Allocation
RT VM	CSA	Existing Filters + RT-Filter	RT-Weigher
Regular VM	Full CPU	Existing Filters	Existing Weighers

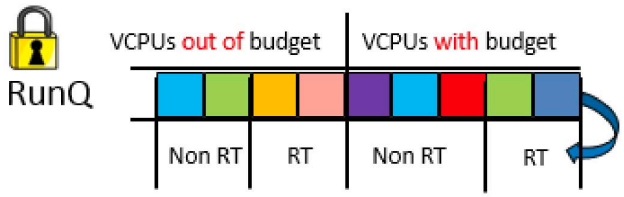


Fig. 1: Run Queue Architecture in RT-Xen 2.1

phase, and use a RT-Weigher in the weighing process. Table I summarizes the different treatment of RT vs. regular VMs in RT-OpenStack. We next detail the design principle behind RT-Filter and RT-Weigher.

RT-Filter: In addition to the existing filters in the Nova scheduler, we implemented a RT-Filter for RT VMs. For each host, the RT-Filter records its total CPU capability (number of available cores), and also the RT VMs with their interface parameters running on the host. Note that regular VMs are ignored in this process, as RT VMs has strict higher priority than regular VMs. When a user creates an RT VM, the RT-Filter performs the compositional scheduling analysis on all hosts one by one, assuming that the RT VM is placed on the considered host. If the output of the compositional scheduling analysis requires more cores than the host's total number of cores, the RT-Filter filters out the host; otherwise recored the host. In the end, RT-Filter returns a set of hosts, each of them can safely accept the newly created RT VM without hurting its real-time performance.

RT-Weigher: After the RT-Filter (and other existing filters) selected a set of hosts for RT VMs, we apply the RT-Weigher to selected the best one to host the created RT VM. Since this paper focuses on the CPU resources, we consider a worst-fit allocation mechanism in terms of remaining real-time CPU capabilities. We first perform the compositional scheduling analysis for RT VMs on each host to calculate the minimal number of cores to schedule existing RT VMs, then calculate the remaining CPU capability as total number of cores minus the minimal required number of cores. Finally, we select the one with most remaining CPU capability to host the created RT VM. Note that RT-Weigher only applies for RT VM request, and also ignores existing regular VMs when performing the analysis. The existing regular VM weighing process is worst-fit based on remaining memory.

In summary, for RT VMs, besides the existing filters, we also use a RT-Filter to filter out hosts in short of CPU resources, and use a RT-Weigher to select a host with most remaining CPU resources to place the VM. Both RT-Filter and RT-Weigher ignores regular VMs as they have lower priority than RT VMs and thus cannot affect RT VMs' performance. For regular VMs, we fall back to existing filters and use the default weighing process to select the host based on remaining memory. A more complex weighing process consider both CPU and memory resources for RT and regular VMs is deferred as future work.

C. Implementation

Recall that OpenStack lacks an adequate resource interface for real-time VMs. We now describe how to specify the real-time resource interface when creating VMs, pass this information to RT-OpenStack, and further pass it to the RT-Xen on selected host.

When creating a RT VMs in RT-OpenStack, we use existing flavors to specify the required number of VCPUs, and pass three pieces of information to the scheduler via a scheduler hint: a total *budget* for all VCPUs, a shared *period* for all VCPUs, and a *rt* flag. When there are multiple VCPUs in a flavor, we evenly distribute the total *budget* among all VCPUs. After a host is selected to host the RT VM, we simply use the information to change the scheduler parameters on the RT-Xen of that host.

For regular VMs in RT-OpenStack, we leverage the same interface as RT VMs for simplicity. We set all regular VMs to use the same *period*, and each VCPU's *budget* equals its *period*. in RT-Xen 2.1, we use the tie-breaking for gEDG scheduling, as a result, all regular VMs are scheduled in a round-robin fashion. Moreover, since all regular VMs' VCPU has full *budget*, they can utilize the remaining CPU resources left by RT VMs as much as possible.

We have implemented RT-Xen 2.1 in Xen 4.1.2 hypervisor. We also extended the RT-Xen tool for including the rt parameters. The RT-Filter and RT-Weigher are implemented in Python and integrated into the OpenStack (version Havana) scheduling framework.

V. Evaluation

In this section, we present our experimental evaluation of RT-OpenStack. We first evaluate the effectiveness of our RT-Xen 2.1 hypervisor scheduler in scheduling RT and regular VMs on a same host. We then conduct a series of experiments to evaluation RT-OpenStack's efficacy in co-hosting RT and regular VMs on a cloud computing platform.

A. Experiment Platform

Our cloud computing testbed contains seven multi-core machines, configured as follows: host 0 is equipped with an Intel 4 core chip with 8GB memory, and works as the controller; host 1 is an Intel i7 4770 4 core machine with 8GB memory; host 2 is an Intel i7 x980 6 core machine with 12GB memory; host 3-6 are Intel i5 4590 4 core machines with 16GB memory each. XenServer 6.2 patched with RT-Xen 2.1 is installed on all machines. On each machine, domain 0 is configured with 1 VCPU, 1 GB memory and is pinned to core 0; the agent VM is configured with 1 VCPU, 3 GB memory

and is also pinned to core 0. The XenServer takes around 200MB of memory on each machine. The remaining cores and memory are used to run the guest VMs. We use the gEDF scheduler with deferrable server on each machine, as it was shown to work best in our previous studies [6]. Within the real-time VMs, we apply the *Litmus^{RT}* [33] patch to Linux 3.10 and use the gEDF scheduler at the guest OS level. All regular VMs run Linux 3.10 as the guest OS. To facilitate predictable real-time performance, we disable dynamic frequency scaling, turbo boost, and hyper-threading so that each PCPU worked at constant speed. All other unnecessary services were turned off during the experiment.

B. Co-scheduling RT and regular VMs on a host

1) RT VM's reservation on a single core: We first run a simple experiment to test the capability of RT-Xen 2.1 to enforce specified CPU resource allocation for an RT VM in face of competing regular VMs on a single host. We boot 1 RT VM and 5 competing regular VMs (VM1 to VM5). They all have 1 VCPU each, and are pinned to a single common core (through cpumask). The RT VM is configured with a *budget* of 4 ms and *period* of 10 ms. All VMs run a CPU busy program to take as much CPU resource as possible. We start with only one RT VM running, then gradually enable the CPU busy program in regular VMs, and record the CPU resources received by the RT VM (via the *xentop* command).

TABLE II: CPU Utilization Test on Single Core

RT VM	40.3%	40.2%	40.2%	40.2%	40.2%	40.2%
VM 1	-	59.5%	29.8%	19.9%	14.9%	11.9%
VM 2	-	-	29.8%	19.8%	14.8%	11.9%
VM 3	-	-	-	19.9%	14.9&	12.0%
VM 4	-	-	-	-	14.8%	12.0%
VM 5	-	-	-	-	-	12.0%
Total	40.3%	99.7%	99.8%	99.8%	99.6%	100%

Table II shows the results. Clearly the RT VM's CPU utilization is not affected by regular VMs, even under stress testing. We also notice that when multiple regular VMs overload the CPU, the remaining CPU resources are distributed evenly among them as expected under our round-robin scheduling policy for regular VCPUs. Another observation is that all CPU utilizations add up to at least 99.5%, which demonstrates the capability of RT-Xen 2.1 to achieve high CPU utilization. For credit scheduler, because there is no distinguish between RT VM and regular VMs, every running VMs would receive equal fraction of the CPU resources, e.g., if RT VM and VM 1-3 are enabled, each VM would receive 25% CPU resources (4 VMs in total). Note that user can configure a VM's credit to get different allocations, but that cannot guarantee the real-time performance of VMs, as demonstrated in previous work [6].

2) Schedulability of RT VMs: The second set of experiments is designed to evaluate the real-time performance of RT VMs under RT-Xen 2.1.

It is also important to show that RT-Xen 2.1 can provide CPU resources to each RT VM at the right time to meet the real-time application's deadlines. We set up this experiment as follows: Each RT VM's contains two real-time tasks, with a period randomly selected from 20 ms to 33 ms, and an execution time randomly selected from 10 ms to 20 ms. The number of VCPUs varies based on the randomly generated task parameters. We iterate through all possible periods for the VCPU(s): for each period, we use the compositional scheduling theory to calculate the desired budget for the VCPU(s). Among getting all the combinations, we pick the one with minimal bandwidth (defined as ratio of budget to period).

We ran the experiments with three PCPUs, and generated 2 RT VMs, the actual total task utilization was 2.03, while the total VCPU bandwidth was 2.93, and they required three full PCPU to schedule them. We again booted up two regular VMs, and configured the cpu test program in sysbench [34] to run in them. The program kept calculating prime numbers, and reported the number of rounds it achieved during a given time. The real-time task and the sysbench task were configured to run for 1 minute. We performed the experiments under three different regular VM workload: 1) both regular VMs are idle; 2) only regular VM1 runs the sysbench; and 3) both VMs runs the sysbench We also repeated the experiment with the credit scheduler.

TABLE III: Schedulability Test on Multi-Core

	Deadline Miss Ratio		Number of Rounds Calculating Pi	
	RT VM 1	RT VM 2	Regular VM 1	Regular VM 2
RT-Xen 2.1	0%	0%	-	-
	0%	0%	1929	-
	0%	0%	1280	1266
Credit	0.01%	0.5%	-	-
	3.4%	15.3%	2596	-
	73.7%	40.7%	1941	1736

As shown in Table III in all the tested cases, RT-Xen 2.1 met the deadlines of the tasks running in the real-time VMs, and evenly distributed the remaining resources for regular VMs. In sharp contrast, under the credit scheduler the real-time VMs experienced deadline misses (0.01% and 0.5%) for both real-time VMs even when there are no interference, and the deadline miss ratio grew to 73.5% for RT VM 1 when there were two regular VMs running.

Summary: On a single host, RT-Xen 2.1 can maintain real-time VMs' performance while allowing regular VMs to utilize the remaining CPU resources.

C. Real-Time Cloud Management

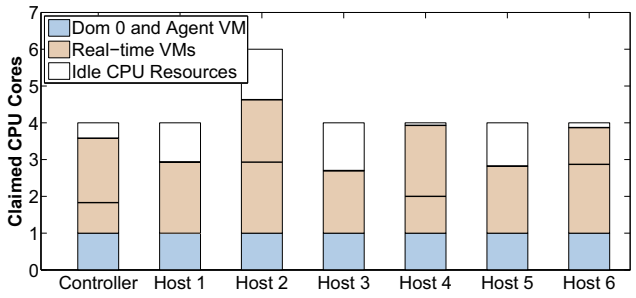
This experiment was designed to evaluate RT-OpenStack on a cluster running RT and regular VMs of seven hosts. In each RT VM, we emulate a cloud gaming server (described in [1]), where there are two real-time tasks: a video encoder and an audio encoder. We generated the configuration for RT VMs in the same way as in Section V-B2, and configured all regular VMs to be a Hadoop cluster to run the standard pi program to test its performance.

We first booted the 7 hosts with RT-OpenStack, then kept creating RT VMs until rejected by the RT-Filter. Each RT VM is configured with 1.5 G memory. 11 RT VMs were created. After that, we kept booting regular VMs with 2 VCPUs and 4G memory each, until one was rejected by the existing filters. 9 regular VMs were booted. We also repeated the same booting sequence using OpenStack for comparison.

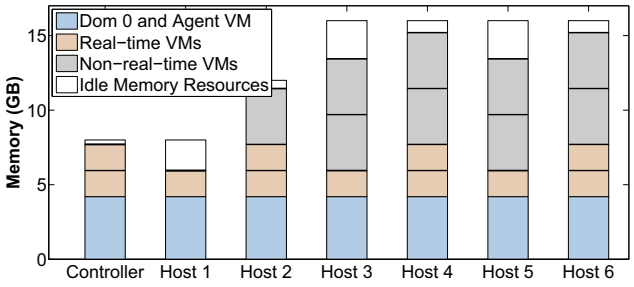
Figures 2a and 2b show the VM allocation scheme under RT-OpenStack. We can see that the RT VMs were evenly

TABLE IV: Deadline Miss Ratio in each RT VM, and Hadoop finish time

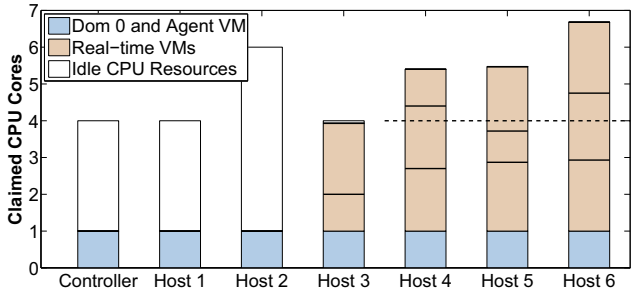
	Deadline Miss Ratio											Hadoop finish time
VMs	RT 1	RT 2	RT 3	RT 4	RT 5	RT 6	RT 7	RT 8	RT 9	RT 10	RT 11	Regular VMs
RT-OpenStack+RT-Xen	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	435 s
RT-OpenStack+Credit	3%	1%	54%	35%	21%	14%	0%	0%	51%	35%	0%	254 s
OpenStack+RT-Xen	9%	0%	0%	0%	2%	0%	0%	0%	41%	11%	0%	–
OpenStack+Credit	37%	31%	61%	13%	75%	29%	30%	36%	73%	47%	32%	314 s



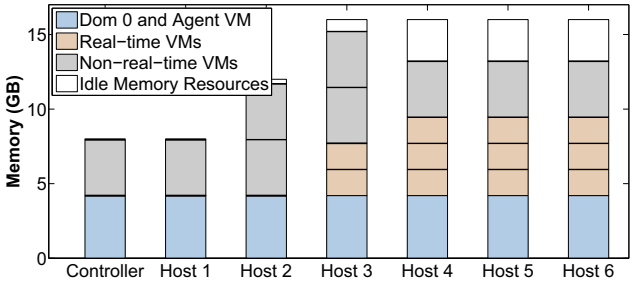
(a) RT-OpenStack: CPU Resources



(b) RT-OpenStack: Memory Resources

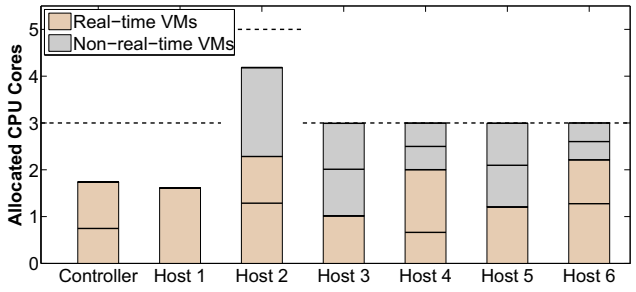


(c) OpenStack: CPU Resources

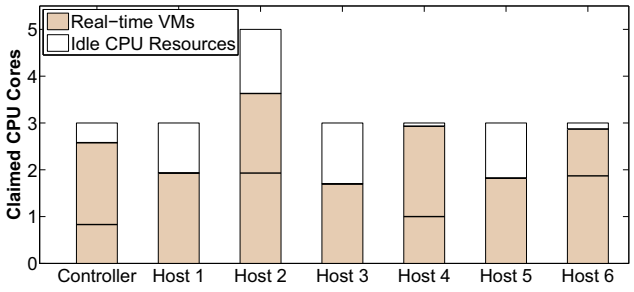


(d) OpenStack: Memory Resources

Fig. 2: VM Allocation



(a) Actual CPU Resource Usage



(b) Claimed CPU Resource Usage

Fig. 3: RT-OpenStack CPU Resource Usage

distributed among the 7 hosts, and the regular VMs were booted on hosts with enough memory to take advantage of the remaining CPU resources. As regular VMs can consume the remaining CPU resources not used by RT VMs under RT-Xen 2.1, we did not show their CPU resource utilization in the figure.

In comparison, Figures 2c and 2d show the allocation by OpenStack using its default filter (does not consider CPU resources) and its worst-fit weigher (based on remaining memory), and the first 11 RT VMs are being packed on hosts 3-6. As a result, these 4 hosts' CPU resources are overloaded.

After all the VMs were ready, we ran the Hadoop workload in the regular VMs, and at the same time started the real-time tasks in the RT VMs. When the Hadoop workload finished, we manually terminated the real-time task in each RT VM and recorded its deadline miss ratio.

Table IV shows the results. The RT-OpenStack + RT-Xen combination experienced no deadline miss in 11 RT VMs, and finished the Hadoop task in 435 seconds; In contrast, using the same RT-OpenStack allocation scheme but with the credit VMM scheduler, 8 out of 11 RT VMs experienced deadline misses, and 2 of them missed more than 50% of the deadlines (RT VMs 3 and 9). However, the Hadoop tasks finished in 254 seconds, which is 3 minutes faster than its completion time with the RT-Xen scheduler. This is because the regular VM receives more CPU resources under the Credit scheduler at the cost of the RT VMs. Interestingly, when using the OpenStack VM allocation scheme with the RT-Xen scheduler, the Hadoop computation makes no progress at all. So we terminated the experiments at 5 minutes and report only the deadline miss ratios in RT VMs. Moreover, four RT VMs experienced deadline misses: we further examined the allocation and found three RT VMs were being allocated on the same host (host 6), saturating its CPU resources. Although RT-Xen can prioritize the CPU resources to RT VMs, due to the allocation scheme, on host 6 there are not enough CPU resources. The CPU overload caused by the VM-to-host mapping by OpenStack led to poor real-time performance despite the real-time hypervisor scheduler. It also explains the freeze of the Hadoop program: RT-Xen strictly prioritize RT VMs over regular VMs, and the Hadoop programs cannot get CPU resources and frozen. This results highlights the importance of real-time VM-to-host mapping even in the presence of real-time hypervisor scheduling. As expected, the OpenStack + Credit combination experienced the worst deadline miss ratios for all RT VMs as a result of both a non-real-time hypervisor scheduling and poor VM-to-host mapping. Also, the Hadoop computation task finished 1 minute later than the RT-OpenStack + Credit combination, due to CPU overloading on hosts 3-6.

Since the Hadoop program takes longer to finish in the RT-OpenStack + RT-Xen combination, it was necessary to test if the CPU resources were fully utilized, i.e., if the Hadoop program consumed as much CPU resource unused by the real-time VMs. We repeated the experiment and recorded each domain's actual CPU consumption for 10 seconds. Figure 3a shows the actual CPU usage of RT and regular VMs, while Figure 3b shows the CPU allocation claimed by the RT VMs according to their resource interfaces. Comparing the actual CPU usage of all VMs with the CPU resources claimed by RT VMs led to the following insights: (1) although the claimed CPU resources almost reached the total CPU capacity, the actual CPU consumption by the RT VM is much less than the claimed ones. This shows the pessimism of the hierarchical scheduling theory that motivates co-hosting real-time VM with regular VMs; (2) on hosts 3 to 6, the actual total CPU utilization had already reached the limit, which means any improvements by the Hadoop program would have affected the real-time performance of RT VMs. On host 2, the actual CPU allocation for non real-time VMs reached 200%, which is the upper limit for 2 VCPUs.

Summary: The combination of RT-OpenStack + RT-Xen can guarantee the real-time performance for RT VMs, while allowing regular VMs to effectively utilizing the remaining CPU resources.

VI. Conclusion

We explore the opportunities and challenges in supporting real-time application in cloud computing platforms, especially in the presence of resource contention from regular VMs. This paper presents RT-OpenStack, a cloud CPU resource manager specifically designed for co-hosting both RT and regular VMs. RT-OpenStack entails three main contributions: (1) integration of a real-time hypervisor and a cloud management system through real-time resource interface; (2) extension of the RT-Xen VM scheduler to allow regular VMs to share hosts without interfering with the real-time performance of RT VMs; and (3) a VM-to-host mapping strategy that provision real-time performance to RT VMs while allowing regular VMs to exploit remaining CPU resources. Experimental results demonstrate that RT-OpenStack can provide latency guarantees for RT VMs in a cloud while achieving high CPU resource utilization by co-hosting regular VMs. While RT-OpenStack represents a step toward supporting real-time applications in the cloud, an area of our future research is extend RT-OpenStack by managing other resources (e.g., storage and network) that can also impact the real-time performance of VMs.

Acknowledgments

This research was supported in part by ONR (grant N000141310800 and N000141310802) and NSF (grant CNS-1329861).

References

[1] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "Gaminganywhere: An open cloud gaming system," in *Proceedings of the 4th ACM multimedia systems conference*, 2013.

[2] "Netflix and stolen time," http://blog.sciencelogic.com/netflix-steals-time-in-the-cloud-and-from-users/03/2011.

[3] "Chaos monkey released into the wild," http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, 2003.

[5] S. Xi, J. Wilson, C. Lu, and C. Gill, "Rt-xen: towards real-time hypervisor scheduling in xen," in *Embedded*

Software (EMSOFT), 2011 Proceedings of the International Conference on.

[6] S. Xi, M. Xu, C. Lu, L. T. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in xen," in *Embedded Software (EMSOFT), 2014 Proceedings of the International Conference on*.

[7] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing compositional scheduling through virtualization," in *Real-Time and Embedded Technology and Applications Symposium, 2012*.

[8] "Rtds based scheduler," http://wiki.xenproject.org/wiki/RTDS-Based-Scheduler.

[9] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, 2008.

[10] "Openstack open source cloud computing software," http://www.openstack.org.

[11] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*.

[12] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*.

[13] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models," in *Real-Time Systems Symposium, 2007. 28th IEEE International*.

[14] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 2007.

[15] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," *Real-Time Systems*, 2009.

[16] S. Groesbrink, L. Almeida, M. de Sousa, and S. M. Petters, "Towards certifiable adaptive reservations for hypervisor-based virtualization," 2014.

[17] E. Bini, G. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *Embedded and Real-Time Computing Systems and Applications, 2009. 15th IEEE International Conference on*.

[18] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *Real-Time Systems Symposium, 2009, 30th IEEE*.

[19] M. Xu, L. T. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill, "Cache-aware compositional analysis of real-time multicore virtualization platforms," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*.

[20] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*.

[21] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the xen hypervisor," in *ACM Sigplan Notices*, 2010.

[22] S. Yoo, K.-H. Kwak, J.-H. Jo, and C. Yoo, "Toward under-millisecond i/o latency in xen-arm," in *Second Asia-Pacific Workshop on Systems*. ACM, 2011.

[23] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms," in *3rd international conference on Virtual execution environments, 2007*.

[24] "Kvm kernel-based virtual machine," http://www.linux-kvm.org.

[25] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical multiprocessor cpu reservations for the linux kernel," in *Proceedings of the 5th international workshop on operating systems platforms for embedded real-time applications (OSPERT 2009), Dublin, Ireland*.

[26] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, "Implementation of compositional scheduling framework on virtualization," *ACM SIGBED Review*, 2011.

[27] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig, "Flattening hierarchical scheduling," in *Proceedings of the tenth ACM international conference on Embedded software*, 2012.

[28] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The xtratum approach," in *Dependable Computing Conference (EDCC), 2010 European*. IEEE.

[29] C. Li, S. Xi, C. Lu, C. Gill, and R. Guerin, "Prioritizing soft real-time network traffic in virtualized hosts," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*.

[30] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, 2011.

[31] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "Vmware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, 2012.

[32] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectivesimplementation for an openstack cloud," in *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*. IEEE.

[33] "Litmus rt: Linux testbed for multiprocessor scheduling in real-time systems," http://www.litmus-rt.org/.

[34] "Sysbench benchmark," http://sourceforge.net/projects/sysbench.