Report Number: WUCSE-2014-004

2014

RT-OpenStack: a Real-Time Cloud Management System

Authors: Sisu Xi, Chong Li, Chenyang Lu, Christopher D. Gill, Meng Xu, Linh T.X. Phan, Insup Lee, and Oleg Sokolsky

Clouds have become appealing platforms for running not only general-purpose applications but also real-time applications. However, current clouds cannot provide real-time performance for virtual machines (VM) for two reasons: (1) the lack of a real-time virtual machine monitor (VMM) scheduler on a single host, and (2) the lack of a real-time aware VM placement scheme by the cloud manager. While real-time VM schedulers do exist, prior solutions employ either heuristics-based approaches that cannot always achieve predictable latency or apply real-time scheduling theory that may result in low CPU utilization. We observe the demand and advantage for co-hosting real-time (RT) VMs with non-real-time (regular) VMs in the same cloud. On the one hand, RT VMs can benefit from the easily deployed, elastic resource provisioning provided by a cloud; on the other hand, regular VMs can fully utilize the cloud without affecting the performance of RT VMs through proper resource management at both the cloud and hypervisor levels. This paper presents RT-OpenStack, a cloud management system for co-hosting both real-time and regular VMs. RT-OpenStack entails three... **Read complete abstract on page 2.**

RT-OpenStack: a Real-Time Cloud Management System

Complete Abstract:

Clouds have become appealing platforms for running not only general-purpose applications but also real-time applications. However, current clouds cannot provide real-time performance for virtual machines (VM) for two reasons: (1) the lack of a real-time virtual machine monitor (VMM) scheduler on a single host, and (2) the lack of a real-time aware VM placement scheme by the cloud manager. While real-time VM schedulers do exist, prior solutions employ either heuristics-based approaches that cannot always achieve predictable latency or apply real-time scheduling theory that may result in low CPU utilization. We observe the demand and advantage for co-hosting real-time (RT) VMs with non-real-time (regular) VMs in the same cloud. On the one hand, RT VMs can benefit from the easily deployed, elastic resource provisioning provided by a cloud; on the other hand, regular VMs can fully utilize the cloud without affecting the performance of RT VMs through proper resource management at both the cloud and hypervisor levels. This paper presents RT-OpenStack, a cloud management system for co-hosting both 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) an extension of the RT-Xen VM scheduler to allow regular VMs to share hosts with RT VMs without jeopardizing 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 among regular VMs. Experimental results demonstrate that RTOpenStack can support latency guarantees for RT VMs, and at the same time let regular VMs fully utilize the remaining CPU resources.

RT-OpenStack: a Real-Time Cloud Management System

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 running not only general-purpose applications but also real-time applications. However, current clouds cannot provide real-time performance for virtual machines (VM) for two reasons: (1) the lack of a real-time virtual machine monitor (VMM) scheduler on a single host, and (2) the lack of a real-time aware VM placement scheme by the cloud manager. While real-time VM schedulers do exist, prior solutions employ either heuristics-based approaches that cannot always achieve predictable latency or apply real-time scheduling theory that may result in low CPU utilization. We observe the demand and advantage for co-hosting real-time (RT) VMs with non-real-time (regular) VMs in the same cloud. On the one hand, RT VMs can benefit from the easily deployed, elastic resource provisioning provided by a cloud; on the other hand, regular VMs can fully utilize the cloud without affecting the performance of RT VMs through proper resource management at both the cloud and hypervisor levels. This paper presents RT-OpenStack, a cloud management system for co-hosting both 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) an extension of the RT-Xen VM scheduler to allow regular VMs to share hosts with RT VMs without jeopardizing 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 among regular VMs. Experimental results demonstrate that RT-OpenStack can support latency guarantees for RT VMs, and at the same time let regular VMs fully utilize the remaining CPU resources.**

I. Introduction

An important advantage to run real-time applications in a cloud is computing power, 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. As a result, such applications require latency guarantees.

Despite the growing demand for running real-time applications in the cloud, however, current clouds provide limited support for such guarantees. 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 in cloud computing.

The lack of systematic support 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 holistic solution to co-host real-time and non real-time (regular) VMs, which entails three main contributions: (1) integration of a real-time hypervisor (RT-Xen) and a cloud management system through real-time resource interfaces; (2) extension of the RT-Xen VM scheduler to allow regular VMs to share hosts without jeopardizing the real-time performance of RT VMs; and (3) a VM-to-host mapping strategy that provisions real-time performance of RT VMs while allowing effective resource sharing by regular VMs. This paper focuses only on the CPU resources; extension to other resources like networks and storage are left to future work. Also note that while this paper focuses on using OpenStack and Xen, the approach can be easily applied readily to other hypervisors and cloud management systems.

The rest of the paper is structured as follows: In Section II we introduce background information on Xen and OpenStack, and their key limitations for supporting RT VMs. We then describe the design and implementation of RT-OpenStack in Section III and present our experimental evaluation in Section IV. After reviewing related work in Section V, we conclude 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.

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 virtual CPUs (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 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].

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 is represented by a resource interface comprising three parameters: *budget*, which decides the amount of time a VCPU is allowed to run per period; *period*; and *cpumask*, a set of the PCPUs on which the VCPU is allowed to run. The current multi-core scheduler in RT-Xen 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 servers and periodic servers. Our experimental results have shown that RT-Xen can provide real-time performance guarantees to the applications running in VMs. Among all the combinations of real-time scheduling policies, global EDF (gEDF) with deferrable server worked best in our evaluation [6].

However, RT-Xen has two drawbacks in supporting RT VMs in a cloud. First, RT-Xen employs compositional scheduling analysis (CSA) [8] to compute the resource interfaces of VCPUs needed to guarantee the real-time performance of applications running in the VMs. While CSA provides the theoretical foundation for providing real-time guarantees on RT-Xen, the resource interfaces computed based on the CSA are often conservative due to the pessimism of CSA. As a result, provisioning CPU resources based on the resource interfaces may lead to significant CPU underutilization (around 60% in our previous experiments [6]). Second, there is no differentiation 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 any latency guarantees. Therefore, if we directly apply the current RT-Xen in a cloud, the host may be under-utilized, and the regular VMs may further reduce the resource utilization.

[1] we use hypervisor and VMM interchangeably

C. OpenStack

OpenStack [9] is a popular cloud management system. It adopts a centralized architecture and consists 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) an admission control scheme for each host to avoid overload; and (3) a 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 specs for 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, which focus on 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, which means if there are 4 PCPUs in a host, the filter can accept up to 64 VCPUs); (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 allocated to a host given the coarse-grained nature of the heuristics used.

VM Allocation: After the filtering process, OpenStack needs to select a host on which 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.

While OpenStack is widely used in genenral 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 and timing granularity needed to achieve real-time performance guarantees. Second, the VM allocation heuristics ignore real-time requirements and allocate VMs based on coarse-grained metrics that are insufficient for provisioning real-time performance guarantees. In the filter stage, OpenStack uses heuristics to decide whether a host is suitable for the VM. In the weighing stage, the current scheme is based on memory and ignores CPU demands that also must be set to satisfy the latency requirements of applications within the VMs.

III. Design and Implementation

A real-time cloud management system for co-hosting real-time and general purpose VMs should have the following characteristics:

- It should provide a real-time resource interface for the VMs that exposes the amounts of resources needed to

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

ensure timing guarantees of the applications running within the VMs, such as the number of VCPUs required by each VM and their real-time parameters (e.g., budgets and periods).

- It should deliver the resources, according to the specification, to the real-time VMs. To achieve this, a real-time VMM scheduler at the host level is required.
- It should perform an appropriate VM-to-Host mapping, which should maintain the schedulability of real-time VMs without overloading the hosts.
- It should be able to co-host general purpose VMs with real-time VMs.
- It should be work-conserving and maintain a high CPU utilization in each host.

We have designed RT-OpenStack, which works at both the single host level and the cloud management level to address these requirements. At the host level, we designed RT-Xen 2.1, an extension of RT-Xen 2.0 that can co-host real-time VMs with regular VMs, maintaining the real-time resource provisioning to real-time VMs, while regular VMs can fully utilize the remaining resources. At the cloud management level, we provide a RT-Filter that forms admission control on each host for real-time VMs, and a RT-Weigher that allocates real-time VMs based on CPU resources.

A. *Co-scheduling RT and regular VMs at single host level*

In RT-Xen the resource interface of a real-time VM is computed using compositional scheduling analysis (CSA) theory [10], which ensures that if the host has sufficient resource to feasibly schedule the VCPUs specified by the interfaces, then all applications running within the VMs are schedulable. We add an "rt" flag for each VM, and re-order the run queue based on this value, as shown in Figure 1.

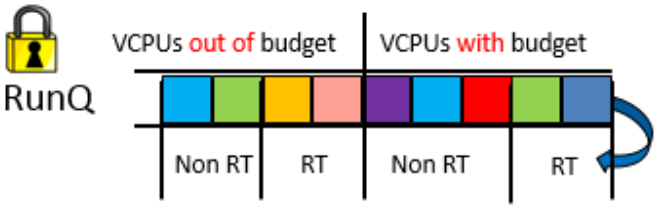


Fig. 1: Run Queue Architecture in RT-Xen 2.1

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.

B. *Co-hosting RT and regular VMs at cloud management level*

Recall that OpenStack lacks an adequate resource interface for VMs, and also the existing Nova-scheduler ignores real-time scheduling analysis. We now discuss how RT-OpenStack addresses each of these concerns:

Resource Interface: In RT-OpenStack each host runs with the RT-Xen 2.1 scheduler, but we still need a method to pass the real-time specification to the cloud manager when creating a VM. One might create various VM template "flavors" with different pre-defined values, but that would be too rigid. In contrast, we use existing flavors which includes the number of VCPUs, and pass three information to the OpenStack via a scheduler hint: whether this VM is a real-time VM or not, the total *budget* for all VCPUs, and a shared *period* for all VCPUs. When there are multiple VCPUs in the flavor, we evenly distribute the *budget* among them.

For a regular VM, a system manager usually does not know the workload characteristics ahead in advance. Since it will not affect the real-time VMs performance, we set the *budget* to be the same as the *period* for regular VCPUs so that they can use the remaining CPU resources whenever available. We also configure the same *period* value for all regular VMs' VCPUs, so they always have the same deadline in EDF scheduling. When multiple VCPUs share the same deadline, RT-Xen 2.1 uses round-robin scheduling among them. As a result, the regular VMs share the remaining CPU resources evenly. Note that for regular VMs, a user can also specify the *budget* to be less than the *period*, which limits the CPU resources to the regular VM. Exploration of this option is left to future work. There are other approaches for integrating general purpose tasks with real-time tasks [11, 12], but they either require the general purpose tasks to follow the same task model as real-time tasks, or they treat general purpose tasks as real-time ones, which further reduces the utilization bound.

RT-Filter: In addition to the existing filters in the Nova scheduler, we implemented a RT-Filter for RT VMs. It acts as an admission controller for real-time VMs on each host. When a real-time VM creation request is submitted, the RT-Filter is triggered on each host. The RT-Filter reads the already accepted real-time VMs' information on the host, and together with the new request, it performs the schedulability test to get the minimal number of PCPUs to schedule those VCPUs. If the required number of PCPUs is larger than the available PCPUs, it rejects the request; otherwise, it accepts the request. Note that the RT-Filter is applied only for real-time VM request, and it considers only the real-time VMs' information when performing compositional scheduling analysis. In this way, we can maintain the real-time VMs performance by not overloading the host, while being able to accept regular VMs to fully utilize the underlying CPU resources.

RT-Weigher: For the VM allocation (weighing) process, we face the problem of considering at least two resources: CPU and memory. Since this paper focuses on the CPU

resources for real-time VMs, we consider worst-fit allocation for real-time VMs based on CPU resource only. We have designed and implemented a RT-Weigher within the Nova scheduling framework. The RT-Weigher works very similarly to the RT-Filter, but returns the remaining CPU capacity on the host. It also considers only real-time VMs when performing compositional schedulability analysis. For the regular VMs, we fall back to the default allocation scheme based on memory.

Table I summarizes the different treatment of RT vs. regular VMs in RT-OpenStack. In summary, we consider only existing real-time VMs' information for RT VMs, and fall back to the default schemes for non real-time VMs.

We have implemented RT-Xen 2.1 in C. We also extended the RT-Xen tool for including the rt parameters. The RT-Filter and RT-Weigher are implemented in Python. Both RT-Xen 2.1 and RT-OpenStack are open source and can be downloaded at https://sites.google.com/site/realtimexen.

IV. Evaluation

In this section, we present our experimental evaluation of co-hosting RT and regular VMs. We first evaluate RT-Xen 2.1 on a single host to demonstrate that regular VMs do not affect the performance of RT VMs. We then conduct a study to demonstrate that RT-OpenStack can satisfy RT VMs' resource requirement when co-locating RT and regular VMs.

A. Experiment Platform

Our 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 i7x980 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}* [13] patch and use the gEDF scheduler at the guest OS level. 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. Impact of regular VMs on real-time VMs

Our previous experiments show that RT-Xen can guarantee the real-time performance on a multi-core platform with moderate overhead. Interested readers can refer to our previous paper [6] for more details. In this paper, we focus on the problem of co-hosting RT VMs with regular VMs.

1) RT VM's reservation on a single core: We first demonstrate that the regular VM cannot affect the CPU resources allocated to an RT VM. We focus on the single-core case, and set up the experiment as follows: We ran the experiment on 1 host, booting 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 and *period* of 10, and each regular VM's *budget* is set to be equal to its *period*. 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 performance 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. Another observation is that all CPU utilizations add up to at least 99.5%, which demonstrates the efficient implementation of RT-Xen 2.1.

We repeated the same experiment with the default credit scheduler, and set each VCPU's cap to 40%. All VCPUs share the same weight. As expected, the CPU resources are equally distributed between the running VMs, and there is no difference among the RT VMs and regular VMs. Due to the relatively large scheduling quantum (30 ms) in the credit scheduler, the numbers obtained via *xentop* varied by +- 2%, so we do not include those numbers here. Note that the system administrator can adjust each VCPU's *weight* to make the RT VM receive a certain amount of resources. However, this requires a global knowledge of all the running domains, and also needs re-adjustment whenever there is a change in the number of VMs. In contrast, when a system administrator allocates a certain amount of CPU resources to a RT VM, it will not change regardless of the number of regular VMs.

2) Schedulability of RT VMs: It is also important to show that RT-Xen 2.1 can provide CPU resources to each RT VM at 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 period randomly selected from 20 ms to 33 ms, and execution time randomly selected from 10 ms to 20 ms. For the underlying VCPU parameters, we iterate through all periods that are less than 30 ms, then use CSA to generate the required budget for each VCPU. After getting all combinations, we pick the one with the minimal total VCPU bandwidth and use it.

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 [14] to run in them. The program keeps calculating prime numbers, and reports 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, and we recorded the results. 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

Table III shows the results. We observe that in all cases, RT-Xen 2.1 can meet the deadline requirements for real-time VMs, and evenly distributes the remaining resources for regular VMs. In sharp contrast, using the credit scheduler 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 up 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 keeping the host utilization high by running regular VMs.

C. Pessimism of Hierarchical Scheduling Theory

We know that hierarchical scheduling theory is pessimistic for RT VMs. This set of experiments aims to quantify that pessimism and see how much CPU resources may be wasted if we ran only RT VMs on a host. A single host is assumed in this experiment. We first assume there is a fixed number of PCPUs, and then keep submitting VM creation requests until the CSA theory rejects the request. We record the total CPU utilization of accepted VMs. For each VM's request, we randomly pick its *period* from 10, 20, ors 40, and also randomly pick its utilization from 10% to 70%. Based on the *period* and utilization, we can calculate its *budget*, using the improved gEDF CSA theory from [15]. Note here that all the results in this subsection are simulation results, but the core algorithm is the same as we used in RT-Filter for RT VMs.

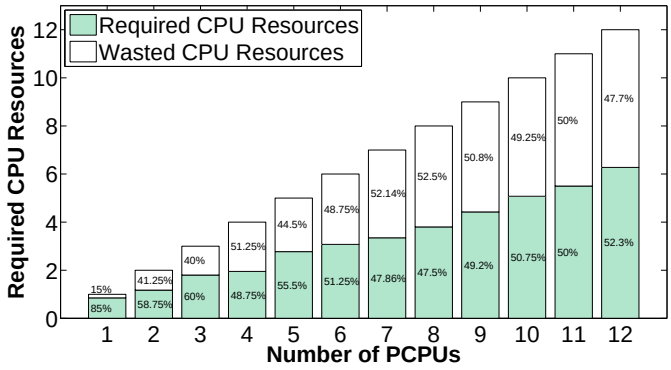


Fig. 2: Simulation Result: RT-Filter

Figure 2 shows the result. On a single core, gEDF becomes a pEDF with 100% utilization, so less than 20% of the resource is wasted. However, once there are more than 2 cores, in all cases more than 40% of the CPU resource is wasted. This clearly demonstrates that scheduling RT VMs alone can greatly under-utilize the CPU resources in a cloud.

Summary: Scheduling only RT VMs can significantly under-utilize the CPU resources, which makes co-hosting RT VM with regular VM desirable.

D. RT-OpenStack Case Study

This experiment was designed to evaluate RT-OpenStack on a cluster. 7 hosts are used. There are two types of VMs running in the cloud, RT and regular VMs.

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 randomly choose each task's period between 20 ms to 33 ms, to emulate different frame rates between 30 fps and 50 fps. Each task's execution time is randomly picked from 10 ms to 20 ms, to represent different games, resolutions, and settings. We applied the *Litmus*^{RT} [13] patch for the RT VM and used the gEDF scheduler to schedule real-time tasks. Compositional scheduling theory was used to generate the VCPU parameters for the RT VM. Here we use the same methodology as in Section IV-C, where we iterate through all the periods for the VCPU and find the one with minimal bandwidth. We configure all regular VMs to be a hadoop cluster, and run the standard pi program to test its performance.

The VM booting sequence was as follows: We first booted the 7 hosts with RT-OpenStack, then kept creating RT VMs until rejected. Each RT VM is configured with 1.5 G memory. Note that each RT VM's task set was different, so their VCPU interfaces also were different. 11 RT VMs were created. After that, we kept booting regular VMs with 2 VCPUs and 4G mem each, until one wasrejected. 9 regular VMs were booted. We also repeated the same booting sequence using OpenStack for comparison.

Figure 3 shows the VM allocation scheme for RT-OpenStack. We can see that the RT VMs were evenly distributed among the 7 hosts, and the regular VMs were booted on hosts with enough memory to take advantage of the remaining CPU resources. Because we configure each regular VM's VCPU *budget* to be the same as its *period* so it can fully use the CPU resources, we did not show their CPU resource in the figure.

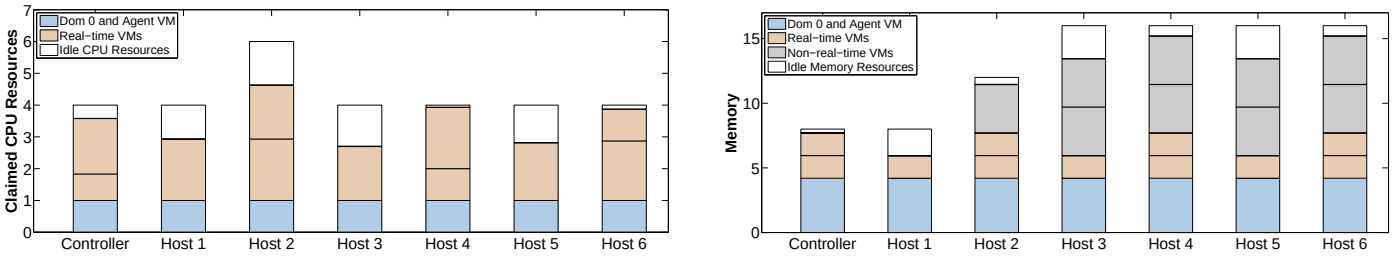
In sharp contrast, Figure 4 shows the allocation by OpenStack. The worst-fit algorithm based purely on memory is used, and the first 11 RT VMs are being packed on hosts 3-6. As a result, these 4 hosts's CPU resources are overloaded.

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 had deadline miss ratio larger than 50%

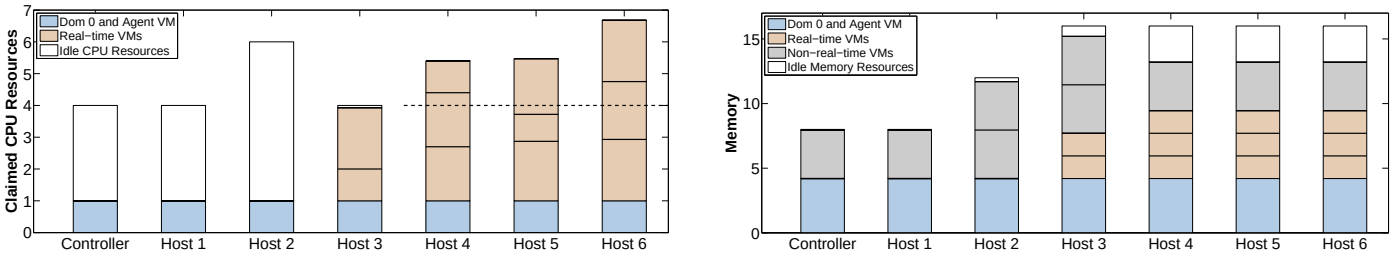
TABLE IV: Deadline Miss Ratio in each RT VM, and Hadoop finish time

	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) CPU Resources (b) Memory Resources

Fig. 3: RT-OpenStack VM Allocation



(a) CPU Resources (b) Memory Resources

Fig. 4: OpenStack VM Allocation

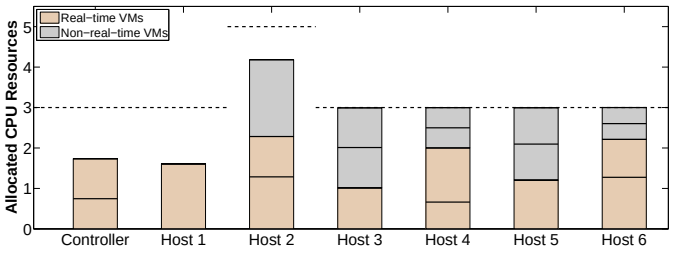


Fig. 5: Actual CPU Resource Usage for RT-OpenStack

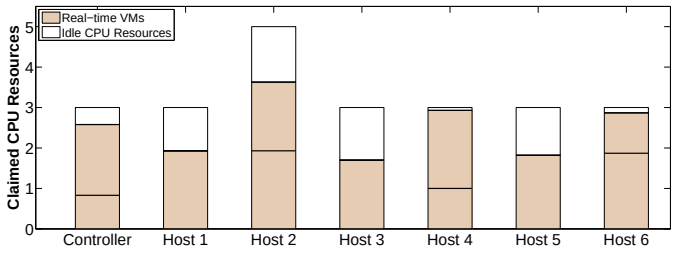


Fig. 6: Actual CPU Resource Usage for RT-OpenStack

(RT VMs 3 and 9). However, the hadoop tasks finished in 254 seconds, which is 3 minutes faster than with the RT-Xen scheduler. This is to be expected as in the Credit scheduler, the regular VMs have more chances to get more resources. When using the OpenStack 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. Four RT VMs experienced deadline misses: we further examined the allocation and found 3 of them weres being allocated on the same host (host 6). This illustrates that 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 OpenStack + Credit combination experienced the worst deadline miss ratios for all RT VMs. This again demonstrated that the default allocation scheme pays no attention to the CPU requirement, and the credit scheduler makes no discrimination of that. Also, the hadoop computation task finished 1 minutes 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, we are also interested whether the host is fully utilized or not. We repeated the experiment, and during it, recorded each domain's actual CPU

consumption for 10 seconds. Figure 6 shows the results. Comparing the actual allocation with the CPU resourced claimed by RT VMs in Figure 3a led to the following insights: (1) although the claimed CPU resources almost reached the limit, the actual CPU consumption by the RT VM is much less than the claimed ones. This further shows the pessimism of the hierarchical scheduling theory, and 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 will affect the real-time performance of RT VMs. On host 2, the actual CPU allocation for non real-time VMs has 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, and let regular VMs use the remaining CPU resources.

V. 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.

A. Single Host

In hierarchical scheduling theory there is a rich body of work for both single-core and multi-core platforms [16, 17, 18, 19, 20, 21, 22, 23, 15, 24]. These approaches cover different aspects of the problem, and differ in the priority scheme used (EDF or RM), scheduling policy used (global or partitioned), server schemes (deferrable server, constant bandwidth server, periodic server, etc), and also the interfaces used in a multi-core environment. However, none of them considers the problem of co-hosting RT VMs with regular VMs. RT-Xen 2.1 is a extension of RT-Xen 2.0, which not only supports co-hosting RT and regular VMs, but also covers most design dimensions and is compatible with most of the theory, such as [22, 18, 15] and their variations. On a one-level scheduling platform, [12, 11] discusses integrating best-effort scheduling into real-time systems. The model they use assumes the same task model for the best-effort tasks. In RT-Xen 2.1, the regular VMs can run any tasks, and we use the average CPU utilization for deciding the parameters. In the future RT-Xen can serve as an open source platform for the community to experiment with different scheduling approaches.

The implementation of hierarchical scheduling has been investigated for various platforms. [25, 26, 27, 28, 29, 30] investigate the problem of implementing hierarchical scheduling in one space (user or kernel). As a result, all scheduling decisions are made by one scheduler. In contrast, RT-Xen performs scheduling at the VMM level, resulting in a clean separation between the two levels of scheduling. Moreover, compositional scheduling theory lets guest domains hide their task-specific details from the underlying platforms, which is suitable for cloud computing. [31, 32, 33] employ heuristics to enhance the default schedulers in Xen. RT-Xen, on the other hand, provides a new real-time scheduling framework to plug-in different real-time schedulers. While the implementation of RT-Xen is specific to the Xen platform, the approach can be easily applied to other virtualization technologies like KVM [34, 35] and micro-kernels [36, 37, 38].

B. Cloud Management Systems

VMWare vCenter [39] maintains each host's utilization between 45% and 81%, and dynamically powers up / 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 load balancing as future work. For open source cloud management systems, the most related work is [40], which also migrates VM to balance the load between hosts. In contrast, RT-OpenStack addresses the VM placement problem to make real-time guarantees for RT VMs.

C. Cloud Applications

Recent work also improves the real-time cloud performance at the application level. For example, [41] focuses on improving the resource allocation for diverse datacenter workloads in hadoop. Our work is complementary to such approaches, as we can let regular VMs use the idle CPU resources.

VI. Conclusion

Real-time applications can benefit from being deployed in cloud, which in turn raises significant research challenges to meet the real-time requirement of such applications. Current clouds cannot provide real-time capabilities for two reasons: (1) the lack of a real-time virtual machine monitor (VMM) scheduler on a single host, and (2) the lack of a real-time aware VM placement scheme by the cloud manager. This paper presents RT-OpenStack, a holistic solution to co-hosting both RT and regular VMs. RT-OpenStack entails three main contributions: (1) integration of a real-time hypervisor (RT-Xen) 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 jeopardizing 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 effective resource sharing by regular VMs. Experimental results demonstrate that RT-OpenStack can support latency guarantees for RT VMs in a cloud, and at the same time let regular VMs fully utilize the remaining CPU resources.

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*. ACM, 2013, pp. 36–47.

[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*, vol. 37, no. 5, pp. 164–177, 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*. ACM, 2011, pp. 39–48.

[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*. ACM, 2014.

[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 (RTAS), 2012 IEEE 18th*. IEEE, 2012, pp. 13–22.

[8] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 30, 2008.

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

[10] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 2–13.

[11] S. Banachowski, T. Bisson, and S. A. Brandt, "Integrating best-effort scheduling into a real-time system," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*. IEEE, 2004, pp. 139–150.

[12] B. B. Brandenburg and J. H. Anderson, "Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors," in *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*. IEEE, 2007, pp. 61–70.

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

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

[15] 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*. IEEE, 2013, pp. 1–10.

[16] 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*. IEEE, 1997, pp. 308–319.

[17] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*. IEEE, 2004, pp. 57–67.

[18] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 129–138.

[19] 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*. ACM, 2007, pp. 279–288.

[20] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," *Real-Time Systems*, vol. 43, no. 1, pp. 60–92, 2009.

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

[22] E. Bini, G. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on*. IEEE, 2009, pp. 294–302.

[23] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*. IEEE, 2009, pp. 377–386.

[24] 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*. IEEE, 2010, pp. 249–258.

[25] J. Regehr and J. A. Stankovic, "Hls: A framework for composing soft real-time schedulers," in *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*. IEEE, 2001, pp. 3–14.

[26] Y.-C. Wang and K.-J. Lin, "The implementation of hierarchical schedulers in the red-linux scheduling framework," in *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*. IEEE, 2000, pp. 231–238.

[27] F. Bruns, S. Traboulsi, D. Szczesny, E. Gonzalez, Y. Xu, and A. Bilgic, "An evaluation of microkernel-based virtualization for embedded real-time systems," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*. IEEE, 2010, pp. 57–65.

[28] T. Aswathanarayana, D. Niehaus, V. Subramonian, and C. Gill, "Design and performance of configurable endsystem scheduling mechanisms," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*. IEEE, 2005, pp. 32–43.

[29] B. Lin and P. A. Dinda, "Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 8.

[30] M. Danish, Y. Li, and R. West, "Virtual-cpu scheduling in the quest operating system," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*. IEEE, 2011, pp. 169–179.

[31] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the xen hypervisor," in *ACM Sigplan Notices*, vol. 45, no. 7. ACM, 2010, pp. 97–108.

[32] S. Yoo, K.-H. Kwak, J.-H. Jo, and C. Yoo, "Toward under-millisecond i/o latency in xen-arm," in *Proceedings of the Second Asia-Pacific Workshop on Systems*. ACM, 2011, p. 14.

[33] 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 *Proceedings of the 3rd international conference on Virtual execution environments*. ACM, 2007, pp. 126–136.

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

[35] 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*, 2009, pp. 15–22.

[36] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, "Im-

plementation of compositional scheduling framework on virtualization," *ACM SIGBED Review*, vol. 8, no. 1, pp. 30–37, 2011.

[37] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig, "Flattening hierarchical scheduling," in *Proceedings of the tenth ACM international conference on Embedded software*. ACM, 2012, pp. 93–102.

[38] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The xtratum approach," in *Dependable Computing Conference (EDCC), 2010 European*. IEEE, 2010, pp. 67–72.

[39] 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*, vol. 1, no. 1, pp. 45–64, 2012.

[40] 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, 2012, pp. 309–315.

[41] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 4.