PARFAIT: A New Scheduler Framework supporting Heterogeneous Xen-ARM schedulers

Jae-Woo Jeong  
Korea University  
Seoul, South Korea  
Email: jwjung@os.korea.ac.kr

Seehwan Yoo  
Korea University  
Seoul, South Korea  
Email: shyoo@os.korea.ac.kr

Chuck Yoo  
Korea University  
Seoul, South Korea  
Email: hxy@os.korea.ac.kr

Abstract—In recent consumer electronics devices, virtualization is widely adopted for diverse reasons such as enhanced reliability, stronger security and better user customizability. Recent CE virtual machines require not only diverse functionalities of GPOS, but also real-time performance of RTOS. However, the current hypervisor-based virtual machine monitors cannot sufficiently support both RTOS and GPOS at the same time. The reason is that the current hypervisor scheduler is biased to only one kind of guest OS and it cannot support heterogeneous schedulers. Therefore, this paper proposes a new scheduler architecture, PARFAIT for both RT and GP guest OSs. Our new scheduler enables to guarantee CPU bandwidth for RTOS and provides fairness among GPOSs using hierarchical structure.

I. INTRODUCTION

Virtualization in mobile and embedded devices is becoming popular. It not only enhances reliability and security by strong isolation of virtualization layer, but also provides better software compatibility and flexible user customizability because it enables to run multiple OSs side by side. Customer electronics devices (or CE devices) such mobile phone or IPTV are trying to adopt virtualization technology because they are dealing with more personally sensitive information, and require more diverse software for better user experience.

CE virtual machine demands not only diverse functionalities of GPOSs (General-purpose operating systems), but also real-time performance of RTOS. Although users in recent CE devices want to have diverse applications across operating systems, many existing CE devices still use RTOS, which has limited functionalities. Consequently, CE virtual machines need to support GPOS so that it can support diverse functionalities from different GPOSs. At the same time, a CE virtual machine has to support RTOS because it has in-nature real-time requirements. In order to timely handle a signal processing for mobile networks or multimedia codecs, RTOS support is an essential requirement in CE virtual machines. With the help of CE virtualization, RTOS can be consolidated with GPOS over the same physical machine.

However, current hypervisor does not sufficiently support both GPOS and RTOS. It is difficult in two folds: 1) RTOS and GPOS require different scheduling policies, and 2) the current scheduler implementation is biased to one kind of scheduler. At first, RTOS requires a resource reservation scheduler because a real-time guest OS requires a static amount of CPU bandwidth for ensuring the schedulability. On the other hand, fair-share scheduler is required to support multiple GPOSs. Secondly, current hypervisor scheduler cannot support both resource reservation and fair-share because the scheduler has to select either one of two.

The goal of this paper is to develop a new scheduler framework supporting both RTOS and GPOS. By introducing a hierarchical scheduling structure, resource reservation scheduler and fair-share scheduler are incorporated within a hypervisor, and each scheduling policy can be preserved.

The paper is organized as following. Section 2 describes related work on supporting RTOS and GPOS. Section 3 presents our design of new scheduler framework. Section 4 provides implementation details for our new framework. Our preliminary results are presented in Section 5. Finally, Section 6 summarizes and concludes the paper.

II. RELATED WORK

A. Limitations of Existing Xen Schedulers

Xen[2] supports BVT, SEDF and Credit schedulers. We classify Xen schedulers into SEDF and the others. SEDF is a resource reservation scheduler, and the others (BVT and Credit) are fair-share schedulers. Resource reservation scheduler tries to allocate a static amount of CPU bandwidth to a specific task, so CPU utilization is guaranteed regardless of the number of tasks. On the other hand, fair-share scheduler tries to evenly distribute CPU bandwidth among all its tasks, and it should be scalable to the number of tasks.

1) SEDF scheduler: SEDF is a resource reservation scheduler which can provide static CPU bandwidth to each guest OS. It assumes that all guest OSs have previously defined scheduling parameter which is composed of period and execution time. The scheduler selects the target guest OS by deadlines. Namely, a guest OS which has the faster deadline has the higher priority, and scheduler always selects the guest OS which has the highest priority.

Although it supports static CPU bandwidth allocation for guest OSs, it has limitations as follows: 1) fairness of SEDF is dependent on scheduling parameter (period, execution). 2) SEDF is not scalable because it requires manual adjustment of scheduling parameters.

It is reported that fairness of SEDF is highly dependent upon scheduling parameter[3]. In order to provide fairness...
among guest OSs, scheduling parameter has to be properly chosen. Otherwise, fairness can be broken or system would be underutilized. For example, let us assume that SEDF tries to fair schedule N guest OSs. Each guest OS should have 100/N% CPU utilization for fair-share. In order to have 100/N% utilization, the bandwidth should be presented with scheduling parameter (period, execution time). If the period is not cleanly divided by N, then the scheduler cannot help choosing another scheduling parameter. Namely, the bandwidth would not be presented with integer-presented scheduling parameter, and it could incur fairness problem.

In addition, SEDF is not scalable because it requires frequent manual operations. Whenever a scheduling parameter is changed, the administrator has to manually modify it via special hypervisor interfaces. Note that the parameter should be changed whenever a VM is created or deleted.

2) BVT, Credit scheduler: BVT[4], [11] and Credit are fair-share schedulers. They evenly distribute CPU bandwidth among all guest OSs. BVT runs with virtual time. Virtual time is accounted by the actual execution time of a guest OS. Thus, virtual time advances only when the guest OS is scheduled to run. The scheduler selects the guest OS which has the lowest virtual time so that all guest OSs can fairly utilize CPU time.

Credit scheduler works based on weighted fair-queuing. All guest OSs receive credit from the hypervisor, and the credit is discharged as much as a guest OS executes. Each guest OS has one of two states UNDER or OVER. When the given credit is fully consumed, and is below zero, then the state is changed from UNDER to OVER. The scheduler selects a guest OS in UNDER state so that it can keep fairness among all guest OSs by credit. Hypervisor recharges credit when all guest OSs are in OVER state. User can control a guest OS by setting the consumption rate of credit, and the total credit.

In order to minimize the latency for I/O operation, BVT uses warp, and credit introduces BOOST state. Both warp and BOOST temporarily prioritize the designated guest OS so that the hypervisor can schedule the guest OS immediately. They are slightly different in that BOOST in credit scheduler cannot specify the amount of prioritization because all I/O-pending guest OSs are in BOOST state, but BVT can specify it by adjusting the warp value.

Neither BVT nor Credit scheduler are appropriate for real-time guest OSs. RTOS demands a fixed amount of CPU bandwidth to ensure the real-time schedulability, so RT-guest OS needs a resource reservation scheduler. However, BVT and Credit schedulers cannot provide a static CPU bandwidth since they only focus on fair-share among guest OSs. Allocated bandwidth using BVT or Credit scheduler decreases as the number of guest OS. As a result, they cannot guarantee the real-time property because of the bandwidth shortage.

B. Time-sensitive application support in General-purpose OS

There are several studies for RT application scheduling inside GPOS. Some of them focus on CPU time reservation [1], [9], and scheduler supports prioritized execution of real-time task. [10] is a OS interface which makes Linux compatible to existing other RTOSs. It enables to run existing RTOS applications over Linux with enhanced latency. They focus on non-preemptive, and fair-share OS scheduler, and tries to achieve fast response of a real-time task, in common.

Proportion Share (PS) algorithm is presented in [7], [12]. They calculate real-time processes weight to meet the requirement of real-time application. [5], [8] show hierarchical scheduling to support the requirement of real-time for multimedia applications support.

They focus on scheduling inside a general purpose OS, but there is a inherent limitation. Because of the invisible overhead inside operating system, real-time task is difficult to be guaranteed to execute at specific time instant. For example, mutual exclusive access to shared resource from GP application can introduce unpredictable latency for RT application. RTOS has resolved problems from operating system by removing thick abstractions in operating systems so that a user can directly handle time-consuming tasks at its own risks. However, it limits by itself because the OS lacks in functionalities, and hard to adopt general purpose applications. On the other hand, in a virtual machine, performance isolation is achieved by more strict hierarchical scheduling. All the time spent by guest OS is accounted by the hypervisor, an RTOS scheduling is not disturbed by any GPOS functions.

III. DESIGN

A. Architecture

This paper proposes a PARFAIT scheduling framework for Xen. In order to incorporate heterogeneous schedulers in Xen, PARFAIT uses hierarchical scheduling structure. Figure 1 presents the overall architecture of our PARFAIT scheduling framework. At top-level, PARFAIT uses a resource reservation scheduler (SEDF), and a fair-share scheduler (BVT) is used at low-level. At the top-level, SEDF partitions the physical CPU bandwidth so that it can guarantee the static amount of CPU bandwidth to RT-guest OSs. At lower-level, BVT schedules GPOS so that all guest OSs can fairly utilize the CPU bandwidth without jeopardizing RT-guest OSs. It is common architecture in hierarchical schedulers; otherwise, (BVT at top-level, and SEDF at low-level) it cannot guarantee the static bandwidth for RT-guest OSs because BVT cannot guarantee a static CPU bandwidth.
A PARFAIT performs scheduling with VCPU of a Guest OS. At top-level scheduler, Dom0 and RT guest OS (such as xeno-uC/OS-II) runs directly with SEDF. In order to schedule GPOSs, PARFAIT introduces a new abstract VCPU which represents all the GPOSs (GP VCPU). When SEDF selects GP VCPU to run, fair-share scheduler (BVT) is invoked. BVT selects the target VCPU to run so that all the GPOS (such as xenoLinux) can fairly utilize the CPU bandwidth.

B. Requirements for PARFAIT

PARFAIT explicitly defines the following five scheduler interfaces as shown in Figure 2: 1) GP VCPU, 2) runqueue, 3) current, 4) idle and 5) system time. Whenever a designated scheduler is changed, then some scheduler interfaces have to be switched properly. For example, if the designated scheduler is changed from BVT to SEDF, then runqueue, current and system time have to be changed from BVT to SEDF specific data structures. GP VCPU is an abstract VCPU data structure representing all the VCPUs of GPOS. In order SEDF to schedule GP VCPU, scheduler private information (sedf_vcpu_info) for GP VCPU is required. Because SEDF requires explicit scheduling parameters (period, execution time) for GP VCPU, PARFAIT provides it on behalf of all GPOSs. At current, PARFAIT statically creates GP VCPU specific data structure when SEDF begins to schedule VCPUs.

Runqueue is a global scheduler specific data structure which maintains the running VCPUs. Because PARFAIT supports two different schedulers, it prepares separate runqueues and switches them whenever the scheduler is changed so that the designated scheduler can use it.

Current is a pointer to the currently running VCPU. Because VCPU also contains scheduler-specific data structure (sched_priv, whose type is void pointer), it has to be switched whenever the scheduler is changed. So, PARFAIT prepares separate current of BVT and SEDF, and switches it properly.

IDLE_BVT is an idle task for BVT. In general, idle task is required commonly to all kinds of schedulers. Because VCPU data structure of BVT is different from that of SEDF, idle task for BVT is separately required.

System time is accounted by NOW. In order to separately account the execution time of GPOS from RTOS, low-level scheduler uses NOW_BVT instead of NOW. PARFAIT maintains NOW and NOW_BVT separately so that each schedulers can use it.

IV. IMPLEMENTATION

We implemented PARFAIT over Xen-ARM hypervisor [6]. Xen-ARM runs with three kinds of guest OS, minios, xenoLinux, xeno-uC/OS-II. Minios is a small operating system for presenting the operation of Xen. xenoLinux and xeno-uC/OS-II are a paravirtualized Linux for Xen, and uC/OS-II, respectively. In this section, we present how the PARFAIT initializes and schedules VCPUs with scheduler interface. We explain how the scheduler interfaces are used at each routine.

A. Scheduler Initialization

PARFAIT constructs hierarchical scheduling structure at scheduler initialization routine. At initialization routine, PARFAIT initializes several data structures for constructing the hierarchical scheduling framework for both schedulers. Namely, all scheduler interfaces of low-level scheduler (BVT) are initialized. Note that many of them would not be used immediately, but initialized for future use.

At first, GP VCPU is created for SEDF. Because SEDF requires a specific scheduling parameter for each VCPU, GP_VCPU also requires a parameter. Although GP_VCPU would not be scheduled at scheduler initialization, it is instantiated for future use. Secondly, runqueues of both schedulers have to be initialized. At initialization, BVT’s runqueue does not have any VCPUs, and IDLE_BVT is inserted at later initialization process. Thirdly, current of BVT is also instantiated although it would not be used at the time. Fourthly, system time of BVT (NOW_BVT) is also initialized to the same value of NOW. Finally, IDLE_BVT is also instantiated, and inserted into BVTs runqueue so that it can utilize physical CPU when it is ready. After this, if GP VCPU is selected by SEDF, then BVT can account time with NOW_BVT.

B. Domain Creation

When a domain is created, VCPU is allocated, in turn. A user can specify a scheduler for the designated domain. If a domain requests fair-share scheduler, PARFAIT sets a VCPU flag, GP_VCPU_NUM. By the flag, PARFAIT initializes the VCPU data structure. Specifically, sched_priv of VCPU is initialized by the scheduler specific data structure. Initially the VCPU is not inserted in runqueue, but it is inserted by a hypercall domain_unpause(). At the time, PARFAIT puts the VCPU into the runqueue of designated scheduler so that the scheduler can pick as a next candidate to run. In addition, GP_VCPU should also be inserted if it (GP_VCPU) is not in the SEDF’s runqueue. During domain creation, system time, IDLE_BVT is not touched.

C. Time accounting of Current VCPU

PARFAIT separates time information of each scheduler so that the scheduler is able to accurately maintain the CPU
usage of local VCPUs. NOW represents the system time of the physical machine. Top-level scheduler (SEDF) uses NOW for accounting the CPU usage not only for RTOS, but also for GPOS. Low-level scheduler uses NOW_BVT for accounting the CPU usage only for GPOS. Note that the top-level scheduler accounts for GPOS time so that it can guarantee the execution time of GP_VCPU. NOW_BVT advances only when the GP_VCPU is scheduled. NOW_BVT is a relative time for low-level scheduler. Low-level scheduler uses NOW_BVT for accounting the execution time of GPOS.

Figure 3 shows how PARFAIT performs time accounting and internal scheduling. Whenever a scheduler is invoked, 1) it updates the CPU usage of the current VCPU. 2) It performs scheduling, and selects the next VCPU to run. 3) It stores the current time instant to next VCPU so that it can calculate the CPU usage at the next time.

In order to calculate the CPU usage for current VCPU, a scheduler (BVT or SEDF) updates execution time of VCPU during the latest scheduling phase. Latest scheduling phase can be calculated by subtracting the last scheduling time instant from NOW. The last scheduling time instant is stored in currents VCPU data structure (runstate.state_entry_time).

The last scheduling time instant is updated after scheduling. Namely, NOW is stored to runstate.state_entry_time of the next VCPU so that the scheduler can calculate the latest scheduling phase at next time. Note that the next VCPU becomes current at the next scheduling.

NOW cannot be used for low-level scheduler because it uses NOW_BVT instead of NOW. In low-level scheduler, latest scheduling phase can be differently calculated. It is calculated by subtracting the last scheduling time instant from NOW_BVT, instead of NOW. Similarly, the next VCPU has to store NOW_BVT, instead of NOW.

Importantly note that in the low-level scheduler, the next VCPU would not be scheduled by BVT. In the case, we should bypass to update runstate of current VCPU because NOW_BVT is not increased during SEDF schedules VCPUs. If not, when SEDF selects GP_VCPU again, the Used2 in Figure 3 is zero because the value of current->runstate.state_entry_time is equal to NOW_BVT. So, we conditionally update runstate for accurate usage accounting as shown in the figure.

D. Internal Scheduling

Because PARFAIT performs hierarchical scheduling, scheduling can be performed twice. If top-level scheduler (SEDF) selects GP_VCPU, then low-level scheduler (BVT) has to selects which VCPU to run. The scheduling decision is returned with next scheduling time. In the case, PARFAIT selects the VCPU from BVT, and selects next scheduling time from minimum value between two results (SEDF and BVT).

E. Scheduler operation

Besides the initialization and scheduling, hypercalls for scheduler operation has to be properly handled. Because there are two different schedulers in PARFAIT, the designated scheduler has to be distinguished for scheduler operation. For example, warp is BVT-specific operation, and SEDF cannot handle it. PARFAIT distinguishes the scheduler by VCPUs flag, GP_VCPU_NUM, and performs proper action for different schedulers.

V. Evaluation

In this section, we present evaluation of PARFAIT. PARFAIT presents a good fairness among GPOSs, and guarantees fixed CPU utilization for RTOS. At first, we present the fairness among GPOSs in PARFAIT. Secondly, we present more complicated scheduling scenarios with several RTOS and GPOS cases in order to present how PARFAIT provides resource reservation for RTOSs as well as fairness for GPOSs. Since PARFAIT uses SEDF as top-level scheduler, simple resource reservation case, which consists of only RTOSs without GPOSs, does not need to be addressed in this paper.

A. Fairness among GPOSs

PARFAIT uses BVT as a low-level scheduler; thus, it fairly distributes CPU bandwidth to all GPOSs. Let us assume that Dom0 runs with top-level scheduler, and it utilizes about 29% of CPU bandwidth. The rest of CPU bandwidth is given to GPOSs. Table I presents the CPU utilization of GPOSs by varying the number of GPOSs. Table shows that PARFAIT fairly distributes CPU bandwidth among GPOSs. All guest OSs perform heavy CPU operation so that it can avoid descheduling by work-conserving mode. As the number of guest OS increases, the CPU bandwidth given to each domain decreases proportionally. Besides, each domain shares CPU bandwidth evenly by PARFAIT. Because PARFAIT schedules GPOSs with low-level scheduler, it provides fairness by BVT.

B. Resource Reservation and Fair-share scheduling through PARFAIT

To viability, we present more complicated examples in this section. As shown in Figure 1 in Section III, PARFAIT schedules two guest OSs (Dom0 and Dom1) with resource reservation scheduler (i.e. SEDF), and two additional guest OSs (Dom2 and Dom3) with fair-share scheduler (i.e. BVT).
Each domain in top-level scheduler requires an explicit parameter (period, execution). Namely, Dom0 and Dom1 have separate parameters so that SEDF can schedule domains with them. For GPOSs (Dom2 and Dom3), PARFAIT provides a shared scheduling parameter because it is scheduled as a single task in top-level scheduler.

Table II shows CPU utilization when PARFAIT schedules domains with different parameters. Parameter of each domain represents the requested CPU bandwidth (=execution/period), and CPU utilization is driven from our experiments. For example, (10, 4) presents 40% of CPU bandwidth.

In case 1, case 2 and case 3, dom0 requests 40% of CPU bandwidth. Dom1 requests different CPU bandwidths (40%, 30%, 20% for case 1, case 2 and case 3, respectively). In case 4 and case 5, Dom0 requests 20% of CPU bandwidth, and Dom0 requests different CPU bandwidths (30%, 20% for case 4 and case 5, respectively). The overall requested CPU bandwidth by Dom0 and Dom1 decreases from 80% to 40% (from case 1 to case 5). Note that Dom2 and Dom3 run with BVT, and do not require specific parameter for each domain.

In all cases, Dom0 and Dom1 can utilize CPU bandwidth almost same with the requested amount. Because Dom0 and Dom1 are scheduled by SEDF, CPU bandwidth should be preserved regardless of Dom2 and Dom3. The rest of CPU bandwidth is utilized by Dom2 and Dom3.

Dom2 and Dom3 are fairly scheduled regardless of given CPU bandwidth. From case1 to case 5, requested CPU bandwidth for GP_VCPU changes from 20% to 60%. In all cases, PARFAIT fairly distributes the CPU utilization between Dom2 and Dom3.

In summary, PARFAIT supports fairness as well as CPU reservation. Regardless of the number of guest OSs, BVT enables to fairly share CPU bandwidth among guest OSs, and it is independent from specific scheduling parameters, which is unfairness-prone in SEDF scheduler.

VI. CONCLUSION

In this paper, we propose PARFAIT, a new scheduling framework for supporting heterogeneous schedulers in Xen. PARFAIT overcomes the biased scheduler implementation of current Xen scheduler. By adopting a hierarchical scheduling structure, PARFAIT incorporates two heterogeneous schedulers so that Xen hypervisor can support both GPOS and RTOS at the same time. It shows a good fairness among GPOSs in various cases. In addition, it preserves requested CPU bandwidth for RTOS accurately. Concisely, it achieves performance isolation not only in terms of resource reservation but also in terms of fair-share as well.

VII. ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by Korea government(MEST) (No. 2010-0029180).

REFERENCES