

Real-time support for Xen

Xen Summit Asia
Nov. 2009

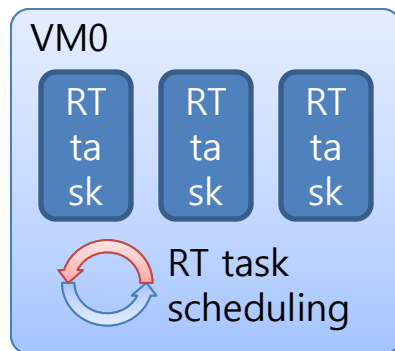
Seehwan Yoo
Chuck Yoo
OS lab, Korea University

Motivational Examples

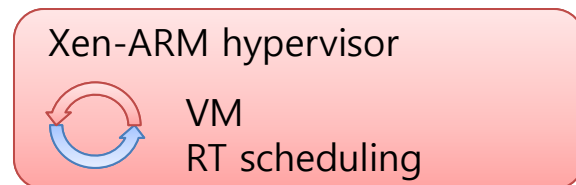
- Accommodate multiple embedded systems over a h/w platform
 - Multi-OS support (UI, apps.)
 - Security, trustworthiness
- Real-time support is essential! (for some embedded systems)
 - E.g.) mobile phone, DTV, etc.
 - Requires not only CPU bandwidth, but also time-bounded processing
- Is real-time possible?
 - **Scheduling is the problem**
 - Why?
 - RT-workload aggregation **changes** required **CPU bandwidth**
 - Shin, Lee, *Compositional scheduling framework with periodic resource model*, TECS 2008

Problem in RT scheduling and VMM

- RT scheduling in physical machine
 - != RT sched. in virtual machine (not aware physical time)
 - != RT sched. in hypervisor (not aware RT tasks)
- Real-time scheduling with aggregated VM's workload



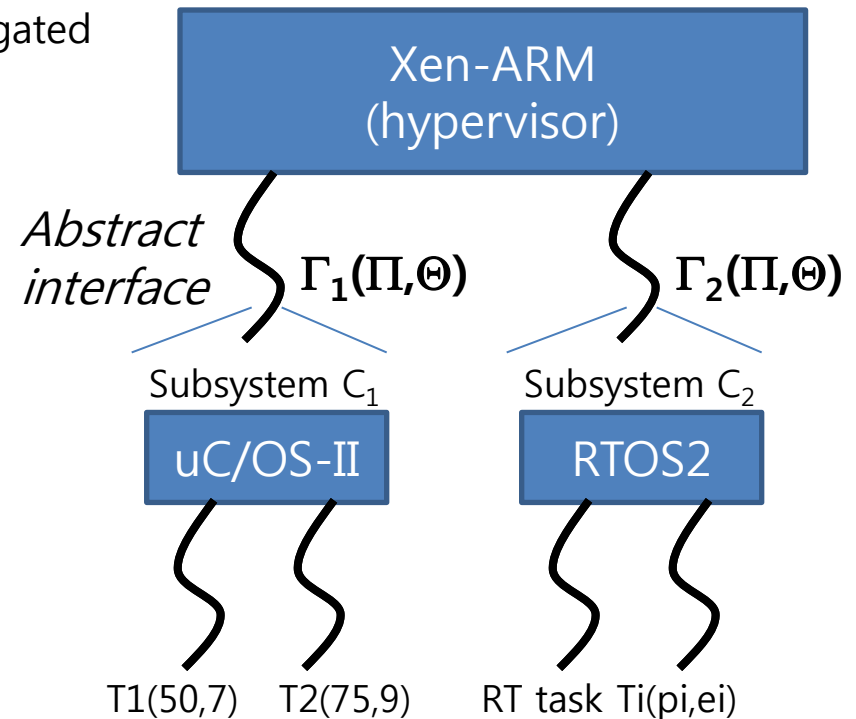
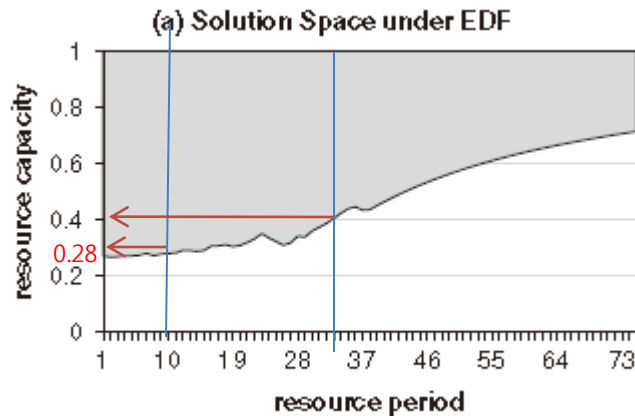
Problem 1.
VM cannot accurately schedule tasks because it runs over virtual time



Problem 2.
Hypervisor is not aware of RT tasks in guest OS

Compositional Scheduling

- Workload aggregation
 - Create aggregated periodic RT task from multiple sub RT tasks
 - Required CPU bandwidth **changes as interface period increases**
- Separately calculate supply/demand of aggregated workload
 - Derive possible schedulable bandwidth with a periodic resource model
- Example (T1: 50,7 T2: 75,9)
 - P: 10, required BW: 28%,
P: 34, required BW: 40%
 - Gray region: Schedulable BW range



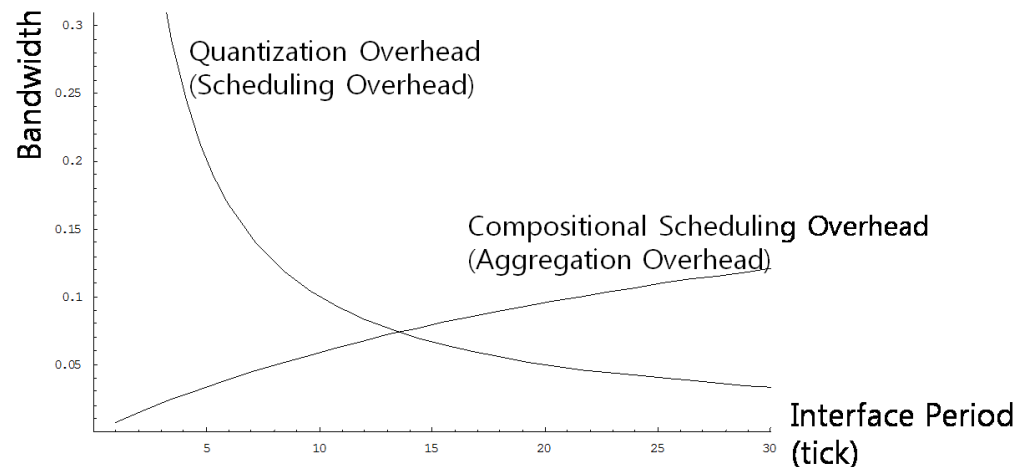
Contribution 1.

Refine scheduling theory

- Quantization effect
 - tick-based scheduling has quantization overhead
 - *Effective bandwidth* : required CPU bandwidth in tick-based scheduling

* To be submitted

- Optimal scheduling period?
 - Minimizes performance overhead
 - Hard to find
(Quantization overhead vs. Compositional scheduling overhead)
 - Find a sub-optimal interface period
 - Algorithm is required



Contribution 2.

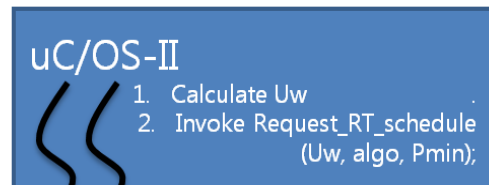
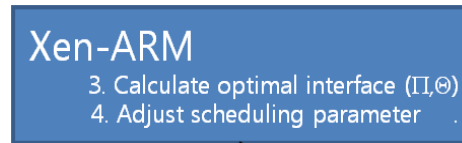
Incorporate theory into VM scheduling

- Interface to specify aggregated RT workload
 - Input to the hypervisor scheduler
 - utilization of the workload,
 - scheduling algorithm (EDF),
 - minimum period in task set
 - Guest provides local (aggregated) workload information
 - New hypercall is introduced
 - `HYPERVISOR_request_RTSchedule` (unsigned long long U_w, char *algo, unsigned int Pmin)
 - Xen calculates the optimal {period, execution time}
 - ➔ Users don't need to find the optimal period, execution time
- Code modification in the source tree
 - Hypercall (arch/arm/xen/hypercalls.S)
 - RT period, execution time decision (common/schedule_realtime.c)
 - Interface with current scheduling (uCOS-ii/kernel/main.c - AppTaskCreate)

Implementation

- Use workload aggregation

- RTOS (such as ucos-ii-xen)
 - 1. calculates U_w
 - 2. provide RT scheduling info (U_w , algo, P_{min})
- Xen-ARM
 - 3. calculates optimal interface
 - 4. adjusts guest scheduling parameter



RT task $T_i(\pi_i, e_i)$

- Instead of finding the interface heuristically;
 - real-time has to be guaranteed

Algorithm 1 Quantized Hierarchical Scheduling Interface Acquisition Algorithm

Require: Input:
 U_w : workload utilization
 Π, Θ : interface period, execution time
 P_{min} : minimum period in workload set
 $AB(k)$: Abstract Bound function
 Ensure: Output:
 (Π_q, Θ_q) : quantized interface period, execution time

```

// Initialization
1:  $EBW = NIL$ 
2:  $\Pi_q = NIL$ 
3:  $\Theta_q = NIL$ 
// Threshold for exhaustive searching
4:  $Th = (1 + 2U_w) / (2U_w(1 - U_w))$ 
5: if ( $Th > P_{min}$ ) then
6:    $Th = P_{min}$ 
7: end if
8: for ( $\Pi = 1; \Pi \leq Th; \Pi++$ ) do
9:    $\Theta_0 = Ceiling(\Pi \cdot U_w)$ 
// Note that  $U_\Gamma \geq U_w$ 
10:  for ( $\Theta = \Theta_0; \Theta \leq \Pi; \Theta++$ ) do
11:     $k = Calculate\_k(P_{min}, \Pi, \Theta)$ 
12:     $AB = Calculat\_AB(k)$ 
13:     $EBW = \Theta / \Pi$ 
// for Schedulable interface,
// Find the minimum EBW interface
14:    if ( $EBW \geq AB$ ) then
15:      if ( $EBW_{min} == NIL \parallel$ 
 $EBW < EBW_{min}$ ) then
16:         $\Pi_q = \Pi$ 
17:         $\Theta_q = \Theta$ 
18:        BREAK;
19:      end if
20:    end if
21:  end for
22: end for
    
```

Our experiences

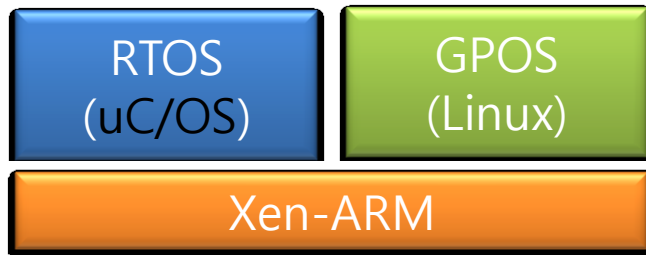
- RT-guest over Xen-ARM
 - ucos-ii-xen
 - Paravirtualize cpu, memory, timer
 - Based on mini-os
 - Compiler change: GCC (from IAR-cc)
 - Can run simple real-time app from GCC toolset
 - MMU protected memory (idempotent mapping)
 - Fixed priority scheduling
 - Has been posted on youtube, xen-arm site
 - XenARM Wiki:
http://wiki.xensource.com/xenwiki/XenARM?action=AttachFile&do=get&target=ucos_ii_xen_on_arm.zip
 - Youtube: <http://www.youtube.com/user/hey mill#p/a/u/2/Vli9zb62Ce0>

RTOS+GPOS

- H/W:
 - Freescale's iMX-21 platform (266Mhz, ARM926)
- S/W:
 - Xen-arm w/ xenolinux 2.6.18
 - SEDF with *quantized compositional scheduling interface*
- Software workload
 - RT: *MiBench-based emulation (for RT emulation)
 - Non-RT: Linux lab. apps. (Mi-bench is used)

*<http://eecs.umich.edu/mibench>

(embedded macro benchmark tool –automotive, telecomm, office, network, security.
we use telecomm –gsm, fft)



Preliminary Result

underway

- GSM Encoding
133 frames
 - ucos-ii-xen (DomU)
 - avg: 3.958(ms),
 - min, max: 1.11, 7.20
 - xenolinux (Dom0)
 - avg. 220ms
- * GSM frame space: 4.25ms
- Difficulties
 - Porting effort
(Benchmark program)
 - LibC has to be omitted
 - Fine-grained time measurement
 - Hypercall overhead at domU
 - Some regs. are not accessible at domU

Conclusion

- Real-time possibility has been presented
- RT scheduling support
 - How to set up scheduling parameters?
- Xen modification
 - RT scheduling interface
- Experiences
 - RT & GPOS
 - MiBench