vMPCP: A Synchronization Framework for Multi-Core Virtual Machines

Hyoseung Kim[*] hyoseung@cmu.edu

Shige Wang[†] shige.wang@gm.com

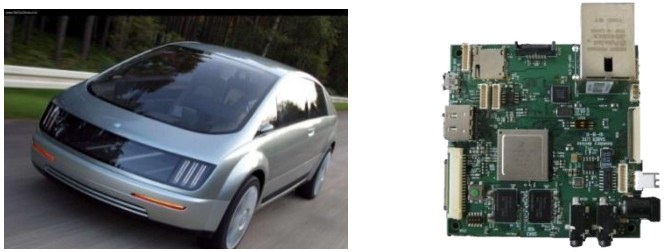
Raj Rajkumar[*] raj@ece.cmu.edu

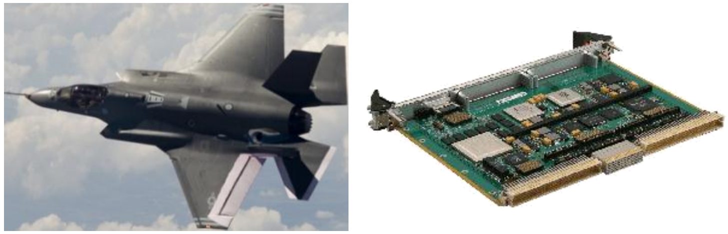
[*] Carnegie Mellon University

[†] General Motors R&D

Benefits of Multi-Core Processors

- **Multi-core CPUs for embedded real-time systems**
    - **Automotive:**
        - Freescale i.MX6 4-core CPU
        - NVIDIA Tegra K1 platform
    - **Avionics and defense:**
        - Rugged Intel i7 single board computers
        - Freescale P4080 8-core CPU





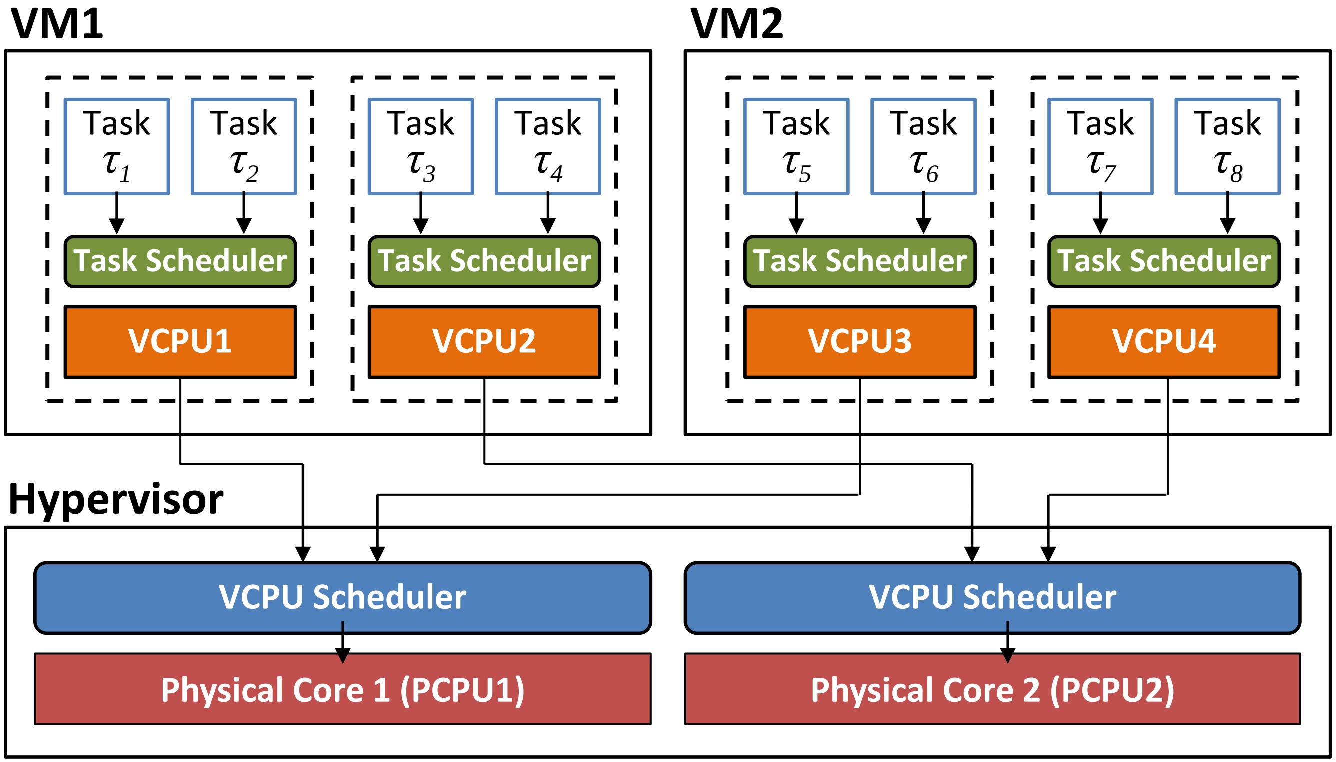
- **Consolidation of real-time applications onto a single multi-core CPU**
    - Reduces the number of CPUs and wiring harnesses among them
    - Leads to a significant reduction in space and power requirements

Virtualization of Real-Time Systems

- **Barrier to consolidation**
  - Each app. could have been developed independently by different vendors
    - Heterogeneous S/W infrastructure
    - Bare-metal / Proprietary OS
    - Linux / Android
  - Different license issues
- **Consolidation via virtualization**
  - Each application can maintain its own implementation
  - Minimizes re-certification process
  - IP protection, license segregation
  - Fault isolation

Virtual Machines and Hypervisor

- Two-level hierarchical scheduling structure
  - Task scheduling and VCPU scheduling



Resource Sharing

- Consolidation inevitably causes the sharing of physical and logical resources
  - Sensors
  - Network interfaces
  - I/O devices
  - Shared memory

Requires mutually-exclusive locks to avoid race conditions

- Increase in processor core count
  - More tasks can be consolidated
  - More resource sharing is expected

We need a synchronization mechanism with bounded blocking times for multi-core real-time virtualization

Previous Work

Context	Synch. protocols	Notes
Multi-core scheduling	MPCP [1] MSRP [2] FMLP [3] MSOS [4]	• Designed for non-hierarchical scheduling • Unbounded blocking time in a multi-core virtualization environment (VCPU preemption / budget depletion)
Hierarchical scheduling	HSRP [5] SIRAP [6] RRP [7]	• Designed for single-core systems • Not extended to multi-core systems • No software mechanism for virtualization

[1] R. Rajkumar et al. Real-time synchronization protocols for multiprocessors. In *RTSS*, 1988
[2] P. Gai et al. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In *RTAS*, 2003.
[3] A. Block et al. A flexible real-time locking protocol for multiprocessors. In *RTCSA*, 2007.
[4] F. Nemati et al. Independently-developed real-time systems on multi-cores with shared resources. In *ECRTS*, 2011.
[5] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *RTSS*, 2006.
[6] M. Behnam et al. SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems. In *EMSOFT*, 2007.
[7] M. Asberg et al. Resource sharing using the rollback mechanism in hierarchically scheduled real-time open systems. In *RTAS*, 2013.

Our Approach

- **vMPCP**: a virtualization-aware multiprocessor priority ceiling protocol
  - Provides bounded blocking time on accessing shared resources in multi-core virtualization
    - Two-level hierarchical priority ceilings
    - Para-virtualization interface
  - VCPU budget replenishment policies
    - Periodic server
    - Deferrable server
  - Optional VCPU budget overrun
  - Implemented on the KVM hypervisor of Linux/RK

KVM

+

Linux/RK

Outline

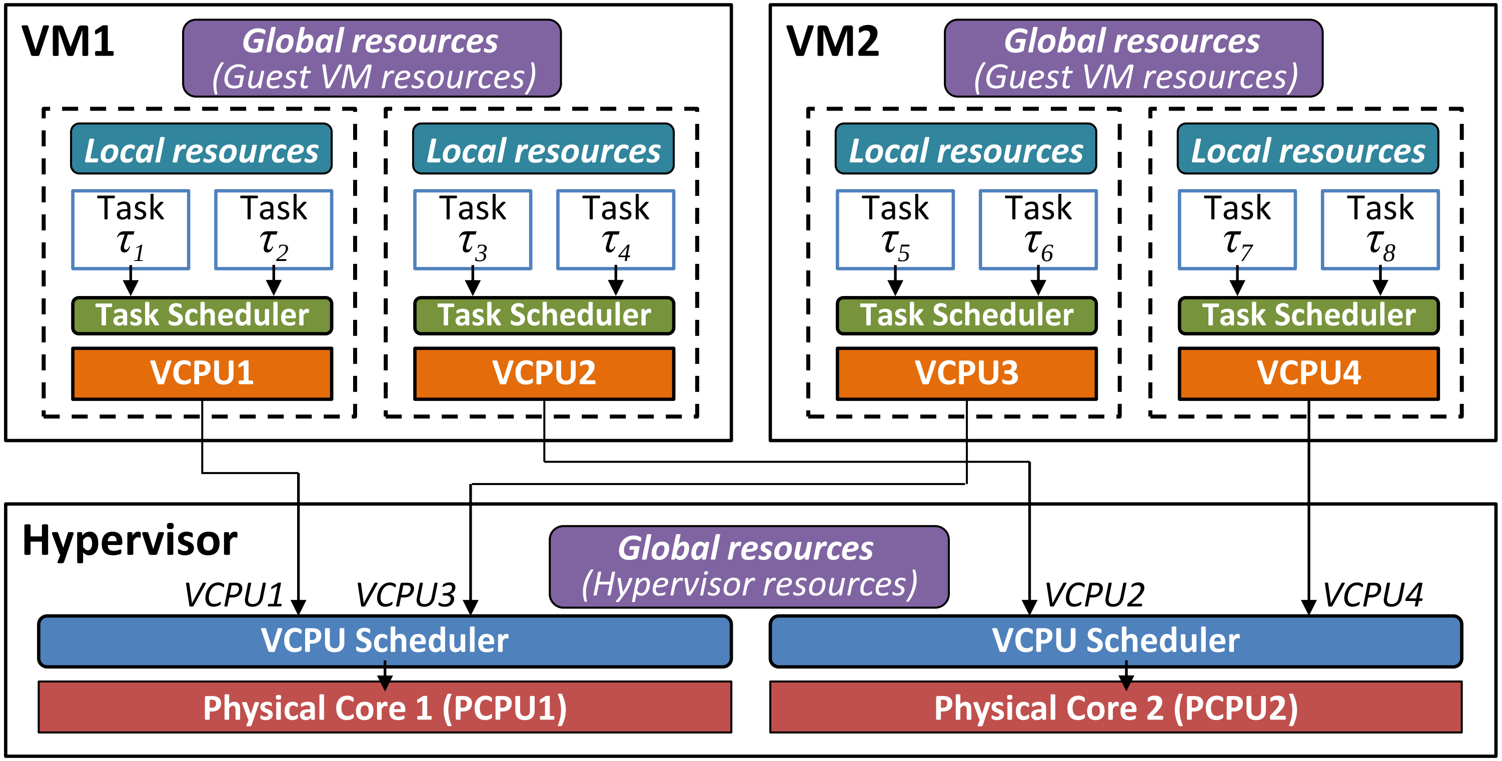
- Introduction
- vMPCP Framework
  - System model
  - Penalties from shared resources
  - vMPCP details
  - Analysis
- Evaluation
- Conclusion

System Model (1)

- Partitioned fixed-priority scheduling for both VCPUs and tasks
- VCPU $v_i: (C_i^v, T_i^v)$
  - C_i^v: Maximum execution budget
  - T_i^v: Budget replenishment period
- VCPU budget replenishment policy
  - Periodic server
  - Deferrable server
- Task $\tau_i: \left(\left(\underline{C_{i,1}, E_{i,1}, C_{i,2}, E_{i,2}, \ldots, E_{i,S_i}, C_{i,S_i+1}}\right), T_i\right)$
  - $C_{i,j}$: WCET of j-th normal execution segment
  - $E_{i,j}$: WCET of j-th critical section segment
  - T_i: Period
  - S_i: The number of critical section segments

<u>Alternating sequence</u> of normal execution and critical section segments

System Model (2)



Local shared resources

Resources shared among tasks on the same VCPU → Local blocking

Global shared resources

Resources shared among tasks on other VCPUs that may be located on other PCPUs → Remote blocking

Penalties from Shared Resources

- **Local blocking**
  - Task waiting on the executions of lower-priority tasks on the same VCPU
- **Remote blocking**
  - Task waiting on the executions of tasks on other VCPUs

Additional timing penalties caused by remote blocking

- Back-to-back execution
- Multiple priority inversions

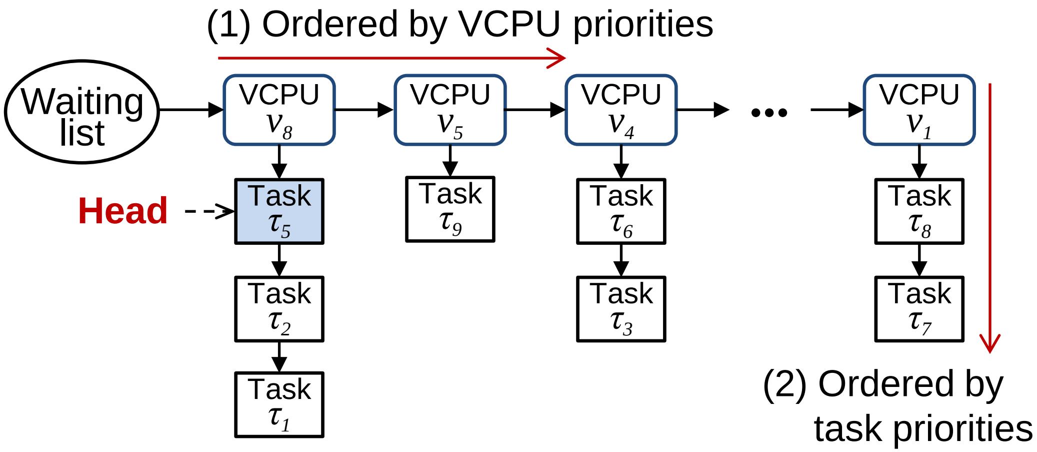
Remote blocking time in a virtualized environment

- Preemptions by higher-priority VCPUs
- VCPU budget depletion

Goal: minimize and bound the remote blocking time
in a multi-core virtualization environment

vMPCP Overview

- **Local shared resource**
  - Follows the uniprocessor PCP
- **Global shared resource**
  - Uses hierarchical priority ceilings (Task-level and VCPU-level)
  - Suppresses task-level and VCPU-level preemptions while accessing a global resource → Reduces remote blocking time
  - Two-level priority queue for a mutex protecting a global resource



No need to compare task priorities in one VPCU with those in other VCPUs
→ Good for different guest OSs (ex, μc/os-ii and Linux)

VCPU Budget Overrun

- vMPCP provides an option for VCPUs to overrun their budgets when their tasks are in global critical sections (gcs's)
  - Allows tasks to complete their gcs's, even though their VCPU has exhausted its budget
  - Pro: reducing remote blocking time
  - Con: more interference to lower-priority VCPUs

Periodic server with overrun

- Obeys the periodic-server's property of having no back-to-back execution

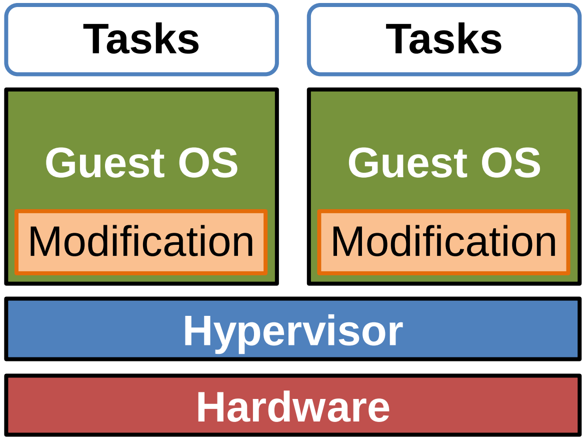
Deferrable server with overrun

- Can overrun more flexibly than a periodic server

Leads to different remote blocking time in analysis

Para-virtualization Interface

- In current virtualization solutions, the hypervisor is unaware of the executions of critical sections within VCPUs
- **Solution: vMPCP para-virtualization interface**
  - What is *para-virtualization*?
    - Small modifications to guest OSs or device drivers to achieve high performance and efficiency
  - To let the hypervisor know the executions of global critical sections within VCPUs
  - Two hypercalls
    - `vmpcp_start_gcs()`
    - `vmpcp_finish_gcs()`



vMPCP Analysis (1)

- **Scope of our analysis**
  - VCPU schedulability
  - Task schedulability
  - Considers four different use cases of vMPCP

VCPU budget replenish policies	With overrun	With no overrun
Periodic server	✓	✓
Deferrable server	✓	✓

vMPCP Analysis (2)

- **VCPU Schedulability**
  - Worst-case response time of VCPU ≤ VCPU period

$$\boxed{W_i^{v,n+1}} = C_i^v + \boxed{O_i^v} + \boxed{B_i^v(W_i^{v,n})} + \boxed{\sum_{v_h \in P(v_i) \wedge h>i} \left\lceil \frac{W_i^{v,n} + J_h^v}{T_h^v} \right\rceil \cdot (C_h^v + O_h^v)}$$

VCPU budget overrun

Blocking time

Higher-priority VCPUs

- **Task Schedulability**
  - Worst-case response time of task ≤ Task deadline

$$\boxed{W_i^{n+1}} = C_i + \boxed{B_i^l + B_i^r} + \boxed{\sum_{\tau_h \in V(\tau_i) \wedge h>i} \left\lceil \frac{W_i^n + J_h + B_h^r}{T_h} \right\rceil C_h} + \boxed{\left\lceil \frac{W_i^n + C_k^v}{T_k^v} \right\rceil (T_k^v - C_k^v)}$$

Local and remote blocking times

Higher-priority tasks in the same VCPU

VCPU budget and budget replenishment period

Outline

- Introduction
- vMPCP Framework
- Evaluation
  - Comparison of different configurations
  - Implementation
  - Case study
- Conclusion

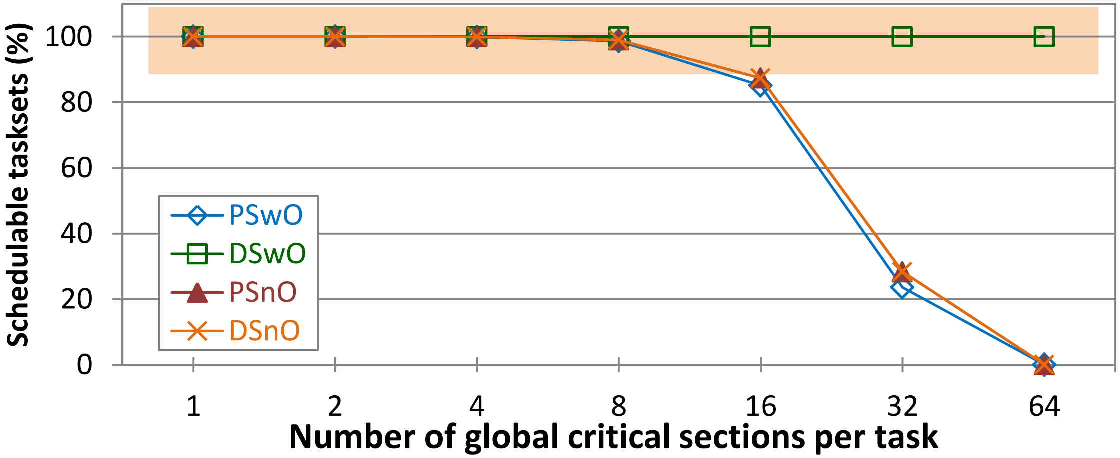
Comparison of Different Configurations

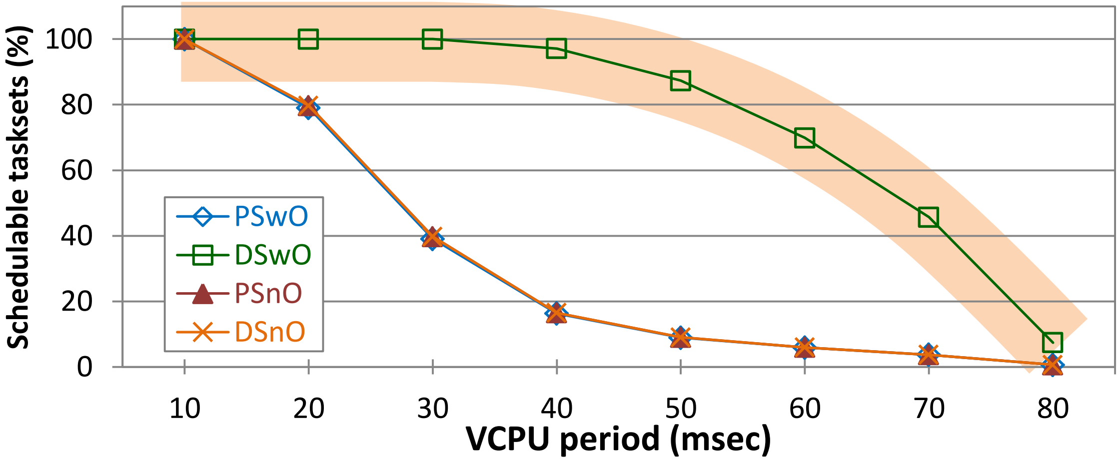
- **Purpose**: to explore the impact of different uses of vMPCP on task schedulability

PSwO	Periodic Server with Overrun
DSwO	Deferrable Server with Overrun
PSnO	Periodic Server with no Overrun
DSnO	Deferrable Server with no Overrun

- **Experimental setup**
  - Used randomly-generated tasksets
  - Metric: the percentage of schedulable tasksets
  - Factors considered
    - Number of global critical sections per task
    - VCPU period
    - Size of a global critical section
    - Utilization of tasks within each VCPU
    - Number of lockers per mutex

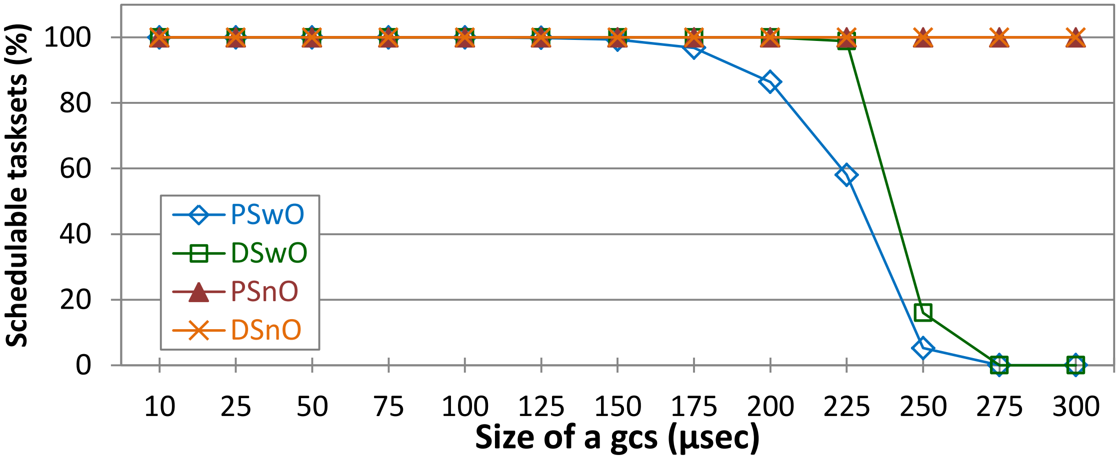
Experimental Results (1)



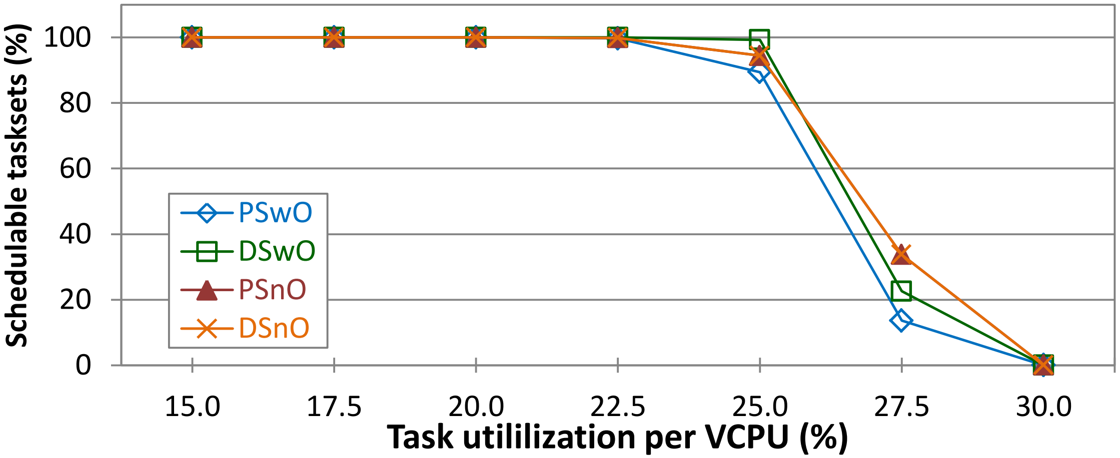


In these two cases, **DSwO** outperforms the other schemes
→ What about other cases?

Experimental Results (2)



The schemes with **no overrun** (**PSnO** and **DSnO**) perform better than the schemes with overrun



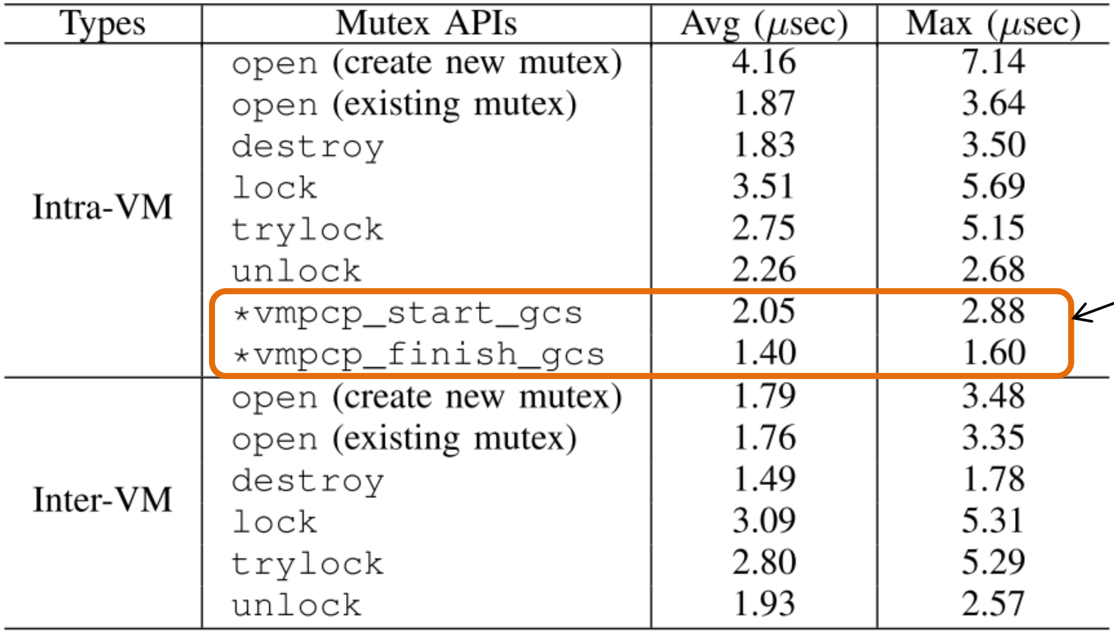
Findings:

(1) There is no single scheme that dominates the others

(2) When overrun is used, a deferrable server outperforms a periodic server

Implementation

- **KVM Hypervisor + Linux/RK**
  - KVM: A full open-source virtualization solution for Linux
  - Linux/RK: Resource kernel implementation based on the Linux kernel
- **vMPCP implementation cost**
  - Target system: Intel Core i7-2600 quad-core 3.4 GHz

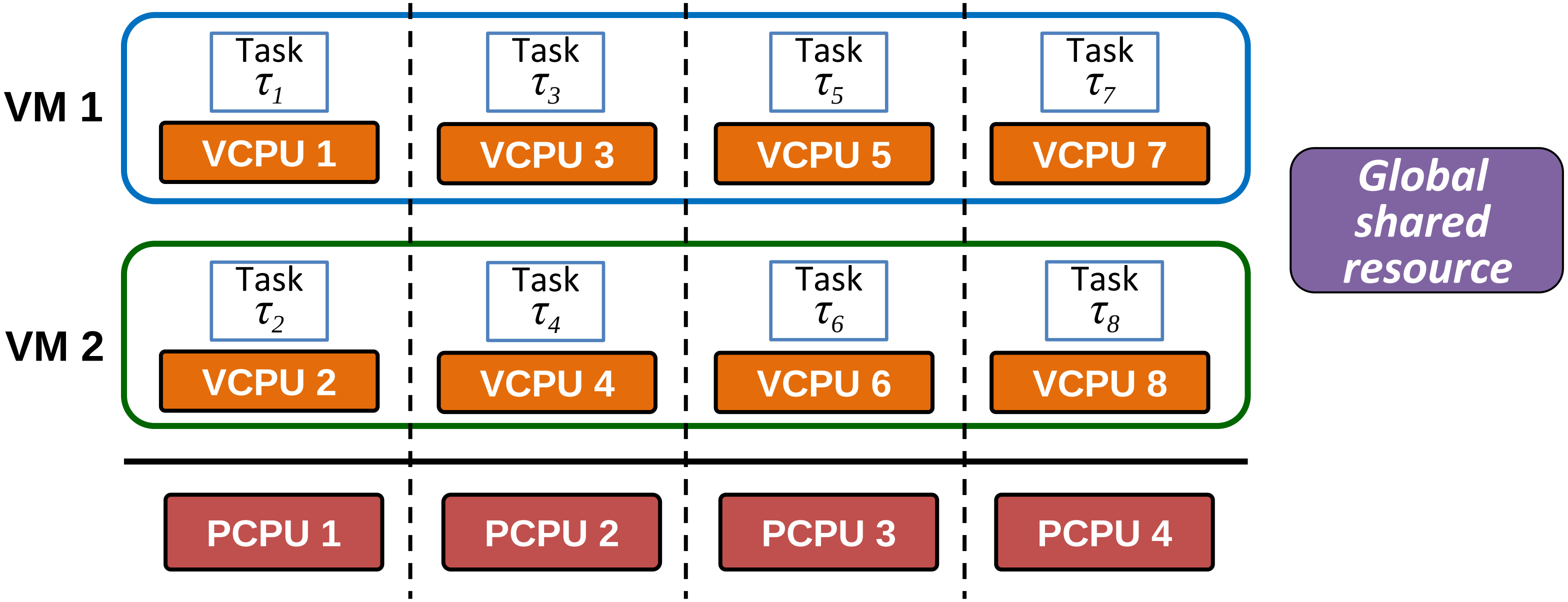


Types	Mutex APIs	Avg (μsec)	Max (μsec)
Intra-VM	`open` (create new mutex)	4.16	7.14
	`open` (existing mutex)	1.87	3.64
	`destroy`	1.83	3.50
	`lock`	3.51	5.69
	`trylock`	2.75	5.15
	`unlock`	2.26	2.68
	$\star$`vmpcp_start_gcs`	2.05	2.88
	$\star$`vmpcp_finish_gcs`	1.40	1.60
Inter-VM	`open` (create new mutex)	1.79	3.48
	`open` (existing mutex)	1.76	3.35
	`destroy`	1.49	1.78
	`lock`	3.09	5.31
	`trylock`	2.80	5.29
	`unlock`	1.93	2.57

Cost for vMPCP para-virtualization

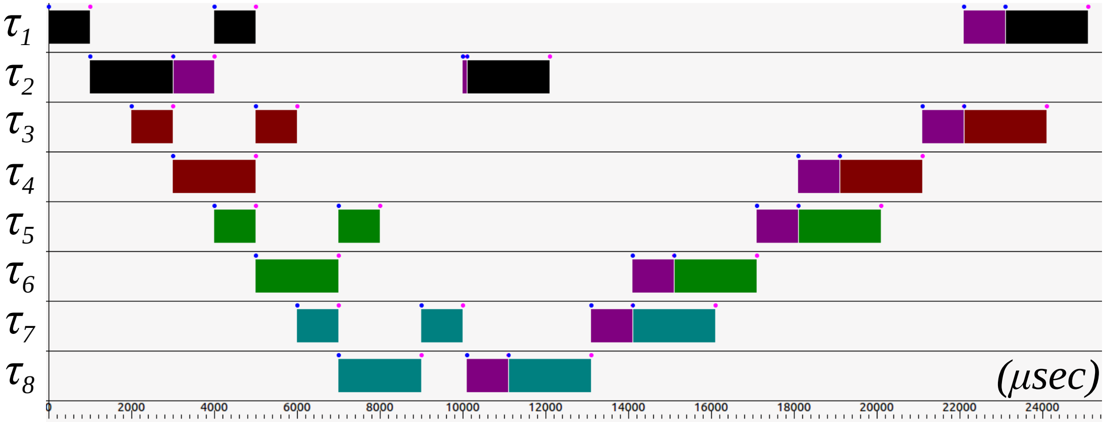
Case Study

- **Purpose:** compare vMPCP against a virtualization-unaware protocol (MPCP)
  - Metric: task response time
- **System configuration**
  - Hypervisor: Linux/RK + KVM
  - Guest OS: Linux/RK
  - VCPU budget replenish policy: deferrable server

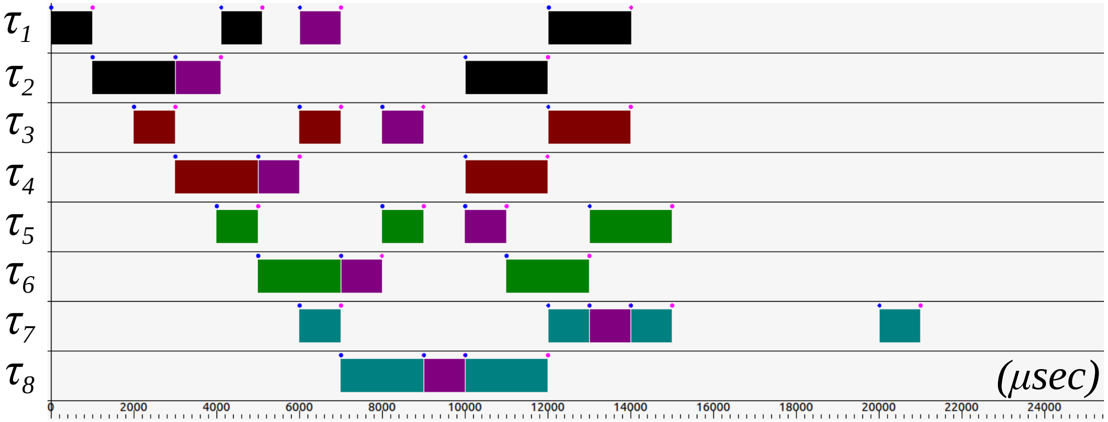


Case Study Results

Virtualization-unaware synchronization protocol (MPCP)



Virtualization-aware synchronization protocol (vMPCP w/ overrun)



vMPCP yields 29% shorter response time on average

Conclusions

- **vMPCP: a synchronization protocol for multi-core VMs**
  - Bounded blocking time on accessing local/global shared resources
    - Hierarchical priority ceilings
    - Two-level priority queue for a mutex waiting list
    - Para-virtualization interface
  - Schedulability analysis and experimental results
    - Deferrable server outperforms periodic server when overrun is used
    - The use of overrun does not always yield better schedulability
  - KVM + Linux/RK: https://rtml.ece.cmu.edu/redmine/projects/rk/
    - In our case study, vMPCP yields 29% shorter task response time compared to a virtualization-unaware synchronization protocol
- **Future Work**
  - Memory interference, compositional framework