Virtual Asymmetric Multiprocessor for Interactive Performance of Consolidated Desktops

Hwanju Kim$^{1,2}$, Sangwook Kim$^1$, Jinkyu Jeong$^1$, and Joonwon Lee$^1$

Sungkyunkwan University$^1$
University of Cambridge$^2$

The 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)
Salt Lake City, Utah, March 1-2 2014
Virtual Desktop Infrastructure (VDI)

- Desktop provisioning

**Dedicated workstations**
- Resource underutilization
- High management cost
- High maintenance cost
- Energy wastage by idle desktops
- Low level of security

**VM-based shared environments**
- High resource utilization
- Low management cost (flexible HW/SW provisioning)
- Low maintenance cost (dynamic HW/SW upgrade)
- Energy savings by consolidation
- High level of security (centralized data containment)
Desktop Consolidation

- Distinctive workload characteristics
  - High consolidation ratio
    - 4:1~15:1 [VMware VDI], 6~8 per core [Botelho’08]
      - Ever-increasing h/w parallelism (multi-core)
  - Multi-layer mixed workloads
    - Diverse user-dependent workloads
    - Multi-tasking (interactive+background) in a consolidated VM

[Botelho’08] Virtual machines per server, a viable metric for hardware selection? (http://itknowledgeexchange.techtarget.com/server-farm/virtual-machines-per-server-a-viable-metric-for-hardware-selection/)
Motivation

• Limited support for interactive performance
  • Existing VMM schedulers give an illusion of symmetric multiprocessor (SMP) to each VM
  • Proportional share-based scheduler used for commodity VMM (e.g., Xen, KVM, VMware)

The size of vCPU = The amount of CPU shares

Virtual SMP (vSMP)

Virtual AMP (vAMP)

Proposed size of vCPU = Proportional share of CPU

Equally contended regardless of user interactions
Goal

• Issue
  • Accurately identifying a set of interactive vCPUs

• Design goals
  • VMM-level approach
    • Identifying interactive vCPUs **unobtrusively** using simple VMM extension
  • No share exchange
    • Distributing CPU shares asymmetrically to sibling vCPUs **within per-VM share budget** for performance isolation
  • Optional guest OS extension
    • Optionally installing lightweight guest OS extension **for further improvement** of interactive performance
Workload Classification

- Previous methods
  - Time-quanta based classification
    [Kim et al., VEE’09]
    - “Interactive workloads typically show short time quantum”
  - Modern interactive workloads show mixed behaviors
  - Multithreaded CPU-bound job shows short time quanta due to inter-thread communication

+ Clear classification between I/O-bound and CPU-bound tasks

[Kim et al., VEE’09] Task-aware virtual machine scheduling for I/O performance
Workload Classification

• Previous methods
  • OS technique
    • User I/O-driven IPC tracking [Zheng et al., SIGMETRICS’10]

+ Identifying a set of tasks involved in a user interaction (I/O)
- Relying on various OS-level IPC structures
  → E.g., socket, pipe, signal

[Zheng et al., SIGMETRICS’10] RSIO: automatic user interaction detection and scheduling
Workload Classification

• Challenges
  • Time-quanta based scheme cannot accurately classify modern desktop workloads
  • VMM cannot access OS-internal IPC structure

• Key idea
  • Tracking background tasks
    • Identifying “background CPU load” before “user I/O”
      • Interactive CPU load is typically initiated by user I/O
      • VMM can unobtrusively monitor user I/O & per-task CPU load
Workload Classification

- Proposed scheme

![Diagram]

- User I/O monitoring => I/O virtualization
- Per-task CPU load monitoring => task tracking technique
  [Jones et al., USENIX’06]

[Jones et al., USENIX’06] Antfarm: Tracking processes in a virtual machine environment
Virtual Asymmetric Multiprocessor

- **vAMP**

- Dynamically adjusting CPU shares of a vCPU according to its currently hosting task

1. Maintaining per-task CPU load during pre-I/O period → Pre-I/O period is set to shorter than general user think time (1 second by default)

2. Tagging tasks that have generated nontrivial CPU loads as background tasks → Threshold can be set to filter daemon tasks that possibly serve interactive workloads

3. Dynamically adjusting vCPU’s shares based on weight ratio (e.g., background : non-background = 1:5)

4. Providing vAMP during an interactive episode → An interactive episode is restarted when another user I/O occurs or is finished if maximum time is elapsed without user I/O
Multimedia Workload Filtering

• Exceptional case
  • Multimedia workloads (e.g., video playback)
    • Can be **misidentified as background workloads** since it continuously generate CPU load without user input

• Key observation
  • Multimedia workloads **generally accompany audio output** [Zheng et al., SIGMETRICS’10, Kim et al., MMSys’12]

• Solution
  • Tracking tasks that access a virtual audio device
    • Excluding audio access in an interrupt context
      • Checking audio Interrupt Service Register (ISR)

[Zheng et al., SIGMETRICS’10] RSIO: automatic user interaction detection and scheduling
[Kim et al., MMSys’12] Scheduler support for video-oriented multimedia on client-side virtualization
Limitation

- An intrinsic limitation of VMM-only approach
  - A vAMP-oblivious OS scheduler
    - Agnostic about underlying vAMP (i.e., all vCPUs are identical)
    - Possibly multiplexing interactive and background tasks on the same vCPU
      - A slow vCPU has higher scheduling latency
      - “Frequent multiplexing” might offset the benefit of vAMP

Example: A scheduling trace during Google Chrome launch

“vAMP might less effective if multiplexing frequently happens”
→ Guest OS can be enlightened to mitigate the adverse effect of multiplexing
Guest OS Extension

• Guest OS extension for vAMP
  • OS enlightenment about vAMP
    • To avoid ineffective multiplexing of interactive and background tasks on the same vCPU → Isolation

• Design principles
  • Keeping VMM OS-independent
    • Optional extension for further enhancement of interactive performance
  • Keeping extension OS-independent
    • No reliance on specific OS functionality
      • Isolating tasks on separate CPUs is a general interface of commodity OSes (e.g., modifying CPU affinity)
    • Small kernel changes for low maintenance cost
Guest OS Extension

- Linux extension for vAMP
  - User-level vAMP-daemon
    - Isolating background tasks exposed by VMM from non-background tasks
    - Small kernel changes that expose background tasks to user

**Isolation procedure:**
1. Initially dedicating \( nr\_fast\_vcpus \) to interactive tasks (i.e., non-background tasks)

2. Periodically increasing \( nr\_fast\_vcpus \) when fast vCPUs become fully utilized
(also periodically checking the end of an interactive episode \( \rightarrow \) stop isolation)

Default \( nr\_fast\_vcpus = 1 \) due to the low thread-level parallelism of interactive workloads
[Blake et al., ISCA’10] Evolution of thread-level parallelism in desktop applications
Experimental Setup

- **S/W**
  - Linux KVM 3.0.0
  - QEMU 1.0: handles I/O requests from guest OS

- **H/W**
  - Intel Xeon X5550 2.67Ghz quad-core processor
    - 8 pCPUs are available w/ hyperthreading enabled
  - 8GB of DDR3 DRAM

- **Measurement methodology**
  - Spiceplay: measures client-side performance
    - Snapshot-based record/replay
      - Robust replay for varying loads
        - Similar to VNCPlay [Zeldovich et al., USENIX’05] and Deskbench [Rhee et al., IM’09]
      - Extension on the SPICE remote desktop client

[Zeldovich et al., USENIX’05] Interactive performance measurement with VNCplay
[Rhee et al., IM’09] DeskBench: Flexible virtual desktop benchmarking toolkit
**Evaluation**

- **Application launch**
  - Background workload
    - Data mining application (freqmine) with 8 threads
  - Weight ratio (background : non-background)
    - vAMP(L)=1:3, vAMP(M)=1:9, vAMP(H)=1:18

**Interactive applications**

- **vAMP improves launch performance by 7~40%**
- **High weight ratio is ineffective because of negative effect of multiplexing**
- **Why did Gimp show significant improvement even without the guest OS extension?**
- **Guest OS extension achieves further improvement of interactive performance by up to 70%**

![Diagram showing evaluation results](image)
Evaluation

- Application launch
  - Chrome vs. Gimp (without guest OS extension)

**Chrome** (Web browser)

→ Many threads are cooperatively scheduled in a fine-grained manner

**Gimp** (Image editing program)

→ A single thread dominantly involves computation with little communication
Evaluation

- Media player
  - VLC media player
    - 1920x800 HD video with 23.976 frames per second (FPS)
  - Mult: multimedia workload filtering

Without multimedia workload filtering, VLC is misidentified as a background task.

vAMP improves playback quality by up to 22.3 FPS, but high weight ratio still degrades the quality.

Guest OS extension achieves 23.8 FPS.
Conclusion

• vAMP
  • Dynamically varying vCPU performance based on their hosting workloads
    • A feasible method of improving interactive performance
  • Assisted by a simple guest OS extension
    • Isolation of different types of workloads enhances the effectiveness of vAMP

• Future work
  • Collaboration of VMM and OSes for vAMP
    • Standard & well-defined API
Thank You!

• Questions and comments

• Source code
  • [https://github.com/VirtualAMP](https://github.com/VirtualAMP)