Coexisting scheduling policies boosting I/O Virtual Machines

Dimitris Aragiorgis, Anastasios Nanos and Nectarios Koziris
{dimara,ananos,nkoziris}@cslab.ece.ntua.gr

Computing Systems Laboratory
School of Electrical and Computer Engineering
National Technical University of Athens

August 30, 2011

Table of Contents

Problem Statement

We focus on:

- busy, **service-oriented** VM containers
- **over-committed** platforms (vCPUs excel physical cores)
- VMs executing **diverse** workloads

We address:

- I/O and especially networking performance
- resources under-utilization of host platforms

We argue that by **altering** the scheduling concept we can

- boost the performace of I/O intensive VMs
- improve I/O utilization of the system
- with little impact on computing performance

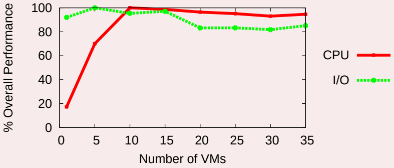
Motivation

Different types of workloads:
(I/O / CPU intensive, Memory bound, low latency, heavy / random I/O)

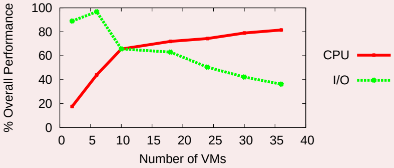
Why scheduling is related to I/O?

- contradicting scheduling demands depending on workload
- more than one domains participate in I/O transactions in VE

Scheduling Effects



(a) exclusive



(b) mixed

Contribution

Alter the scheduling concept:

- Do **NOT** rely on a "one size fits all" scheduler
- Allow co-existing scheduling policies
- Partition resources (cores)
- Match VMs to the corresponding scheduler (depending on workload)

Why Co-existing scheduling policies are attractive?

- Unified schedulers are complex
- Schedulers tailored to specific workload needs are **lightweight**
- Facilitate reuse of existing scheduling algorithms

Achievements: (18 CPU + 18 I/O VMs in 8-core platform)

- GigaBit link **saturation** vs. 38% utilization
- sustain more than 80% of CPU utilization

Table of Contents

The Xen VMM - skb flow in PV

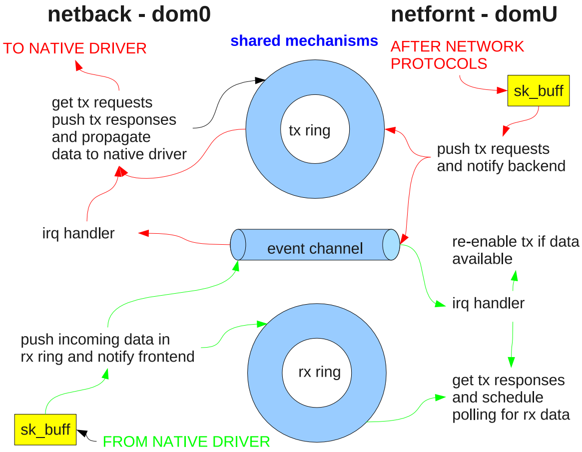


Figure: netfront-netback interaction using I/O rings and events

The Xen VMM - Scheduling Concept

Credit Scheduler Basic Characteristics

- priority and credits based
- 30ms time-slice and 10ms accounting period

Shortcomings

- VM yields the processor before accounting $\Rightarrow$ no credits debited $\Rightarrow$ advantage over others that run for a bit longer
- BOOST vCPUs are favored $\Rightarrow$ CPU-bound domains get neglected in case of fast I/O
- CPU bound VM exhaust its time-slice $\Rightarrow$ I/O service gets stalled

CPU pools

- a group of physical cores
- a specific scheduler

Table of Contents

Evaluation infrastructure

Testbed

VM container:
8-core
Intel Xeon X5365 @ 3.00 GHz

⇔
⇔
4xGigaBit

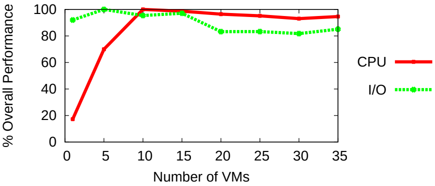
Client:
4-core
AMD Phenom @ 3.2 GHz

Measurement tools

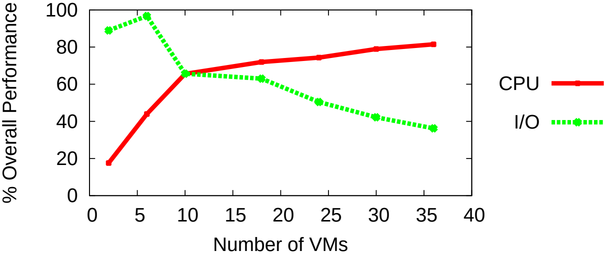
Linux generic tools **emulate** *intensive* applications:

1. I/O (stream/ftp): from memory direct to network
   - i.e. `dd if=/dev/zero | netcat`
2. CPU: from memory to memory
   - i.e. `bzip2 -c /dev/shm/file.img > /dev/null`

Default Setup - Vulnerabilities



(a) CPU or I/O VMs



(b) CPU and I/O VMs

Our Monitoring Tool

Purpose

A tool that can measure the scheduling effect on I/O performance.

Design and Implementation

Concept: Measure the time spent between event occuring and handling in network split driver model. **How**: Inserting time-stamps of wall time.

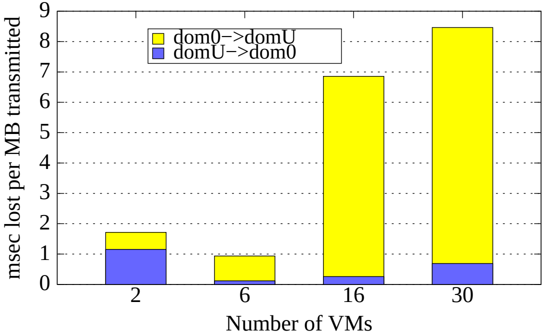
Additional modules

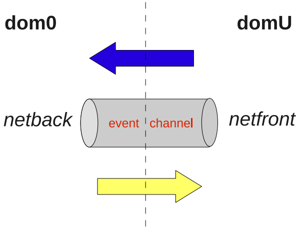
- trigger to start/stop monitoring and initialize data
- cookies to gather all timestamps (cookies) from each domain.

What do we eventually measure?

avg. msec lost per MB transmitted

Default Setup - Monitoring Tool results





- yellow $\gg$ blue
  - dom0 wakes up more frequently due to more I/O requests $\Rightarrow$ able to batch work
- overall time lost increases along with overcommitment
  - CPU VMs exhaust their time-slice $\Rightarrow$ I/O VM get stalled
  - driver domain gets scheduled in and out repeatedly

Decoupling dom0 from VMs - Our *no-op* Scheduler

Purpose

Dedicate a physical core to a vCPU and never preempt it, thus guarantee maximum computing power and responsiveness.

Usage

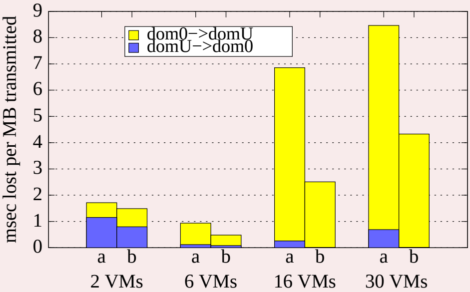
Busy domains as dom0 or stubdomains, real time domains

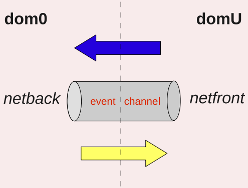
SMP-aware Design and Implementation

- track down all available cpus in the pool
- every CPU is either occupied (by a vCPU) or not
- attach every newly created vCPU to a non-occupied CPU
- insert a vCPU in a waiting list if all CPUs are occupied
- replace a destroyed vCPU with the first on the waiting list

Decoupling dom0 from VMs - 2 pool Setup

Monitoring Tool results



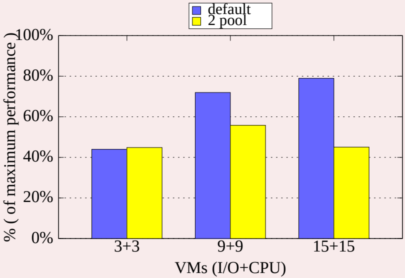


(a) default Setup
(b) 2 pools Setup

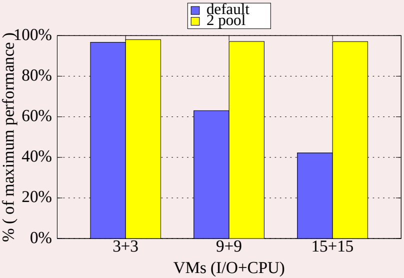
- domU → dom0 (blue) eliminated
  - dom0 never gets preempted
- dom0 → domU (yellow) decreases
  - dom0 processes requests more efficiently ⇒ more data rate available
  - domU get notified more frequently

Decoupling dom0 from VMs - 2 pool Setup

Resources Utilization



(c) CPU Overall Performance



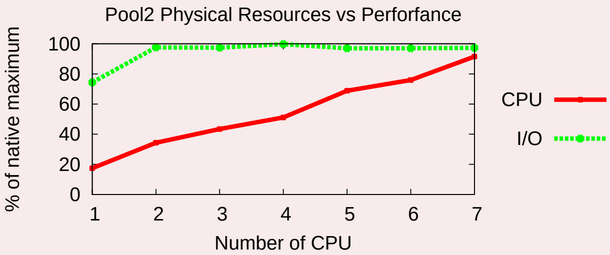
(d) I/O Overall Performance

Remarks

- I/O vCPUs get boosted more frequently
  $\Rightarrow$ CPU vCPUs get neglegted
  $\Rightarrow$ CPU performance decreases

Decoupling dom0 from VMs - 2 pool Setup

Resources destribution in pool containing the VMs

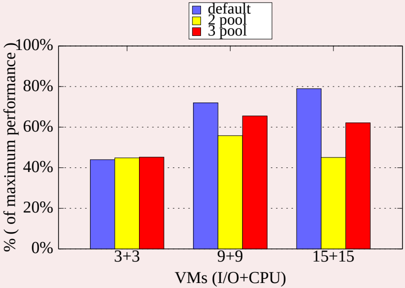


Remarks

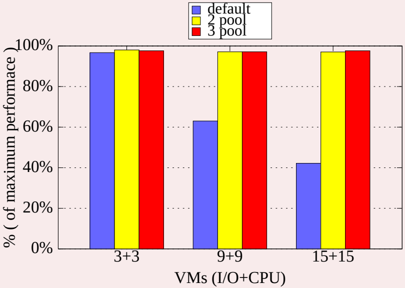
- CPU performace decreases along with the resources reduction
- only two physical cores needed to saturate 1Gbps

Decoupling I/O and CPU VMs

3 pool Setup



(e) CPU Overall Performance



(f) I/O Overall Performance

Decoupling I/O and CPU VMs

Misplacement effect on Individual Performance

	Misplaced VM	All other
CPU	-17%	-1.3%
I/O	+4%	-0.4%

VMs running **similar** workloads should use the **same** scheduler. Overall performance degradation if a VM is misplaced.

Remarks

- Gigabit saturation vs. 38% utilization
- less than 20% decreased CPU performance
- on many-cores the negative effect on CPU intensive VMs should be negligible
- we take the first step towards co-existing scheduling policies and prove it can benefit resources utilization and overall system performance

Co-existing Scheduling Policies - Abstract Schematic

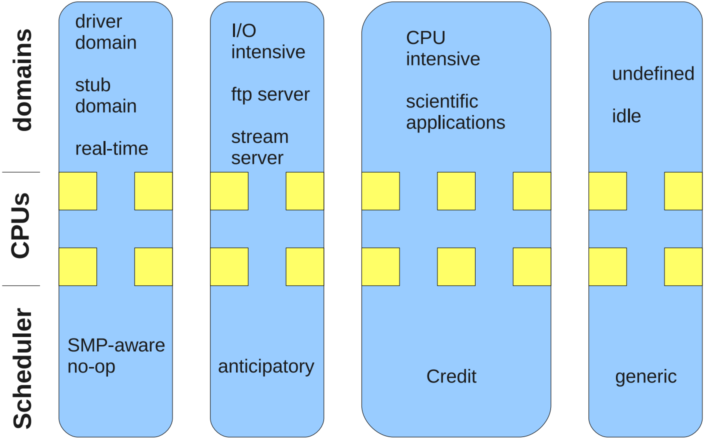
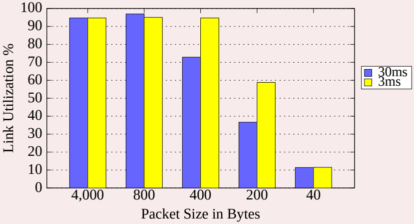


Table of Contents

Discussion for Credit Optimizations for I/O service

Timeslice allocation: 3ms vs. 30ms



Can apply to a random I/O workload (e.g. busy web server)

Anticipatory concept

Concept:

Take advantage of the propability of transmitting or receiving data in the near future.

Implementation:

Make use of multi-hierarchical priority set

and adjust priority when a vCPU wakes up, sleeps or gets credits debited

Purpose:

sustain the vCPU in boost state a bit longer

Contribution

- prove that co-existing scheduling policies benefit I/O:
  - GigaBit link **saturation** vs. 38% utilization
  - sustain more than 80% of computing performance
    - improves in many-cores
- targeted VE:
  - over-commited, service-oriented VM containers
  - VMs with multiple types of workload (intensive or not)

Future Work

- Implement the anticipatory scheduler
- Experiment with scheduling algorithms
- make use of advanced hardware:
  a) multiple NICs
  b) 10GbE network adapters
  c) many-core platforms
  d) multi-queue and VM-enabled NICs
  e) hardware accelerators
- Deploy benchmarks and real-world scenarios
- Implement a profiling system for dynamic system partitioning and VMs placement

Thanks!