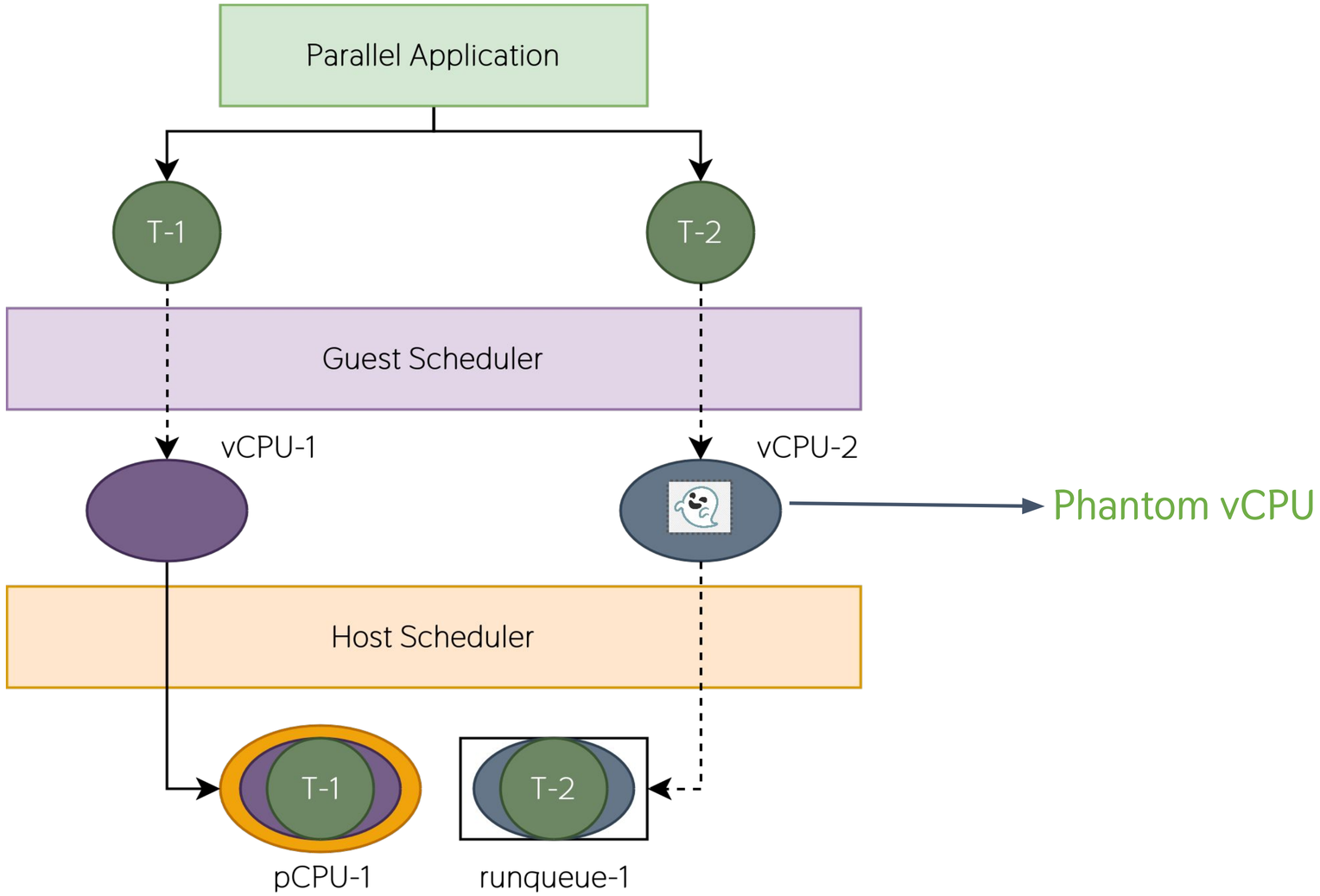
A case for using para-virtualized scheduling information with sched_ext schedulers

Himadri CHHAYA-SHAILESH
Jean-Pierre LOZI
Julia LAWALL

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Context: Dual level of task scheduling for VM workloads



Context: Para-virtualized scheduling information for guest Parallel Application Runtimes (PARs)

- **Problem**
  Degree of Parallelization (DoP) is determined by the number of vCPUs in the guest, but one or many vCPUs might be phantoms on the host

- **Impact**
  Suboptimal performance of guest parallel applications, especially when overload occurs on the host

- **Solution / Policy**
  Aggregate scheduling information about vCPUs on the host, and use it to adjust the DoP in the guest
  i.e. **curr_dop = prev_dop - avg_phantoms + avg_idle_pcpus**

- **Implementation**
  - Target PAR: libgomp — GCC's implementation of OpenMP
  - Implemented by modifying the OMP_DYNAMIC interface

Experiment set-up

- **Host**
  Intel Xeon Gold 5220, 1-socket, 18 cores, 2 threads/core (36 pCPUs), 96 GB, Debian-testing
- **Guest**
  1-socket, 36 cores, 1 thread/core (36 vCPUs), 50 GB, Debian-12
- **Host scheduler**
  EEVDFS from linux-kernel v6.11-rc4 (from the sched_ext tree)
- **Guest scheduler:**
  EEVDFS from linux-kernel v6.6.16 (from the stable tree)
- QEMU v7.2.2 (Debian 1:7.2+dfsg-7)
- libgomp from GCC-12

VM workload: UA (input class B) from NPB3.4-OMP

- Categorized as a benchmark for unstructured computation, parallel I/O, and data movement
- Consists of three major loops, implemented with a total of 38,768 internal barriers
- Worker threads can spin (OMP_WAIT_POLICY=*active*) or block (OMP_WAIT_POLICY=*passive*) upon reaching a barrier while waiting for other threads

Spinning vs Blocking

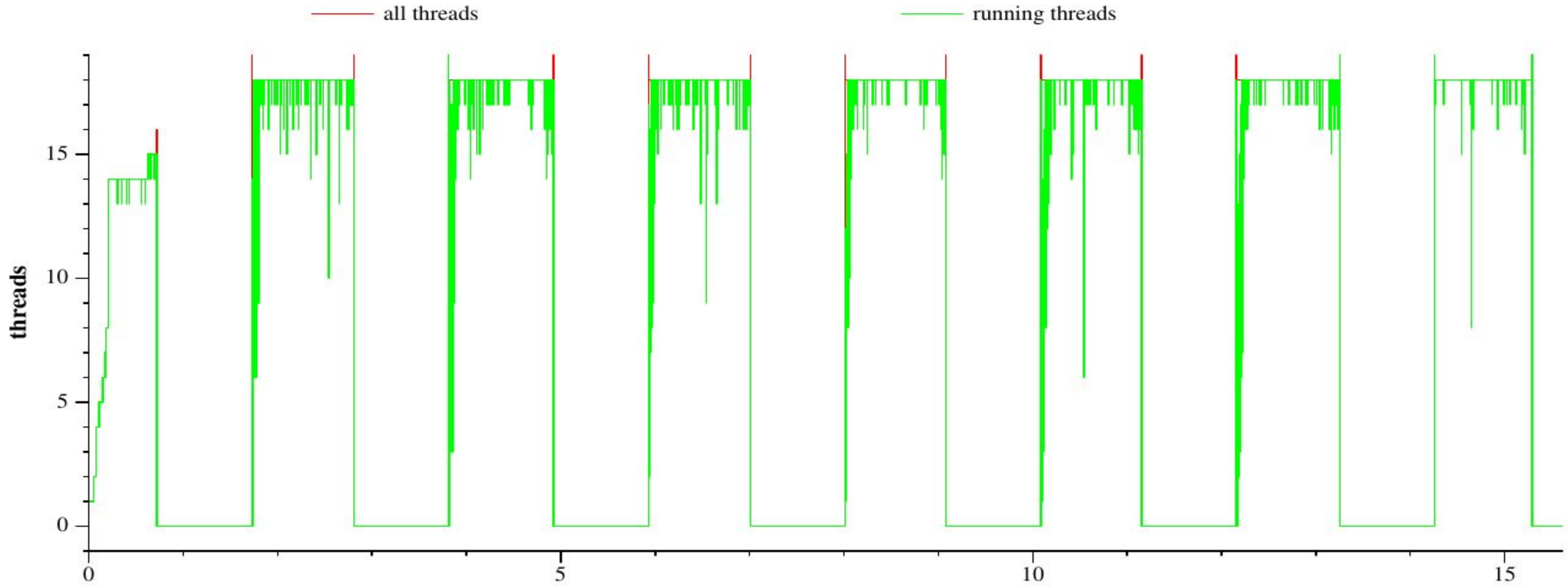
- Spinning is faster than blocking if the host is non-overloaded, i.e. there are no phantom vCPUs
  - OMP_WAIT_POLICY=active: 9.68 ± 0.04 seconds (1.80x)
  - OMP_WAIT_POLICY=passive: 17.43 ± 0.09 seconds

- The performance of spinning suffers greatly in comparison to blocking if the host is overloaded i.e. there are phantom vCPUs
  - On a periodically overloaded host,
    - OMP_WAIT_POLICY=active: 19.95 ± 0.5 seconds (0.48x)
    - OMP_WAIT_POLICY=passive: 22.43 ± 0.09 seconds (0.78x)

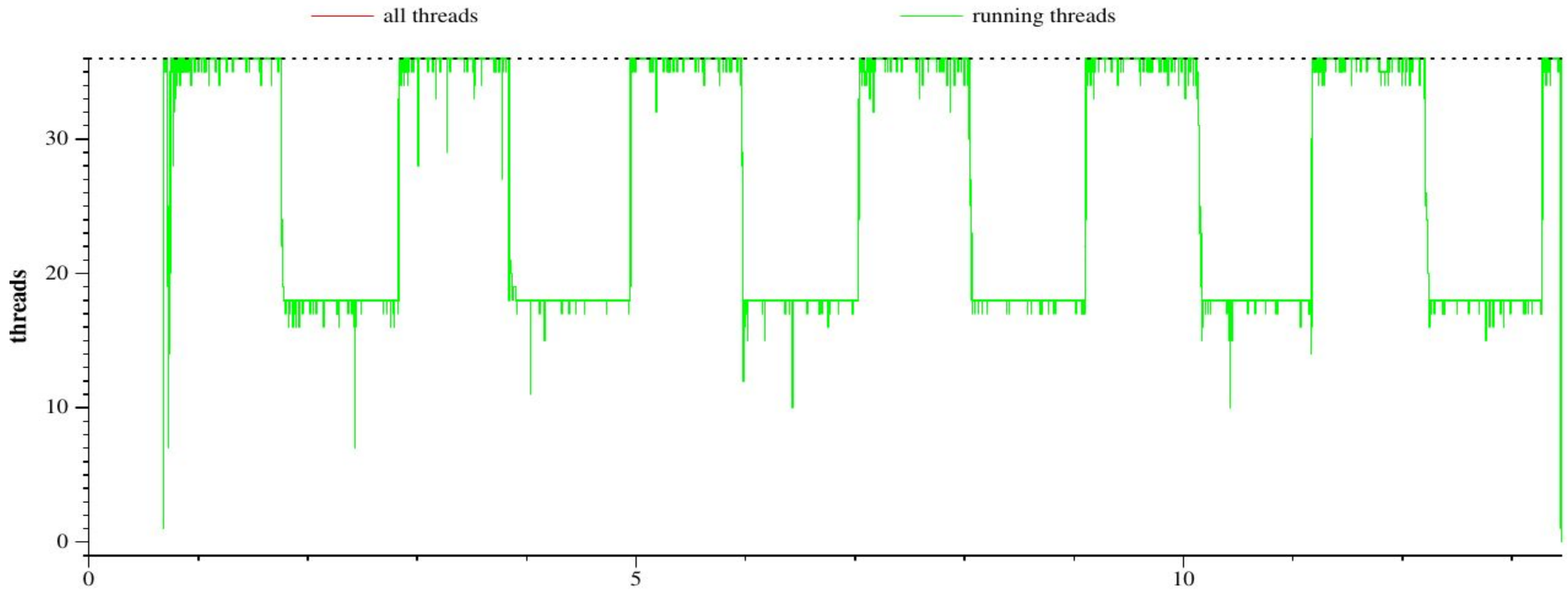
- Degradation in spinning performance increases with increase in number of phantom vCPUs

- TL;DR We want to use spinning while minimizing the number of phantom vCPUs

Periodically overloaded EEVDFS-host



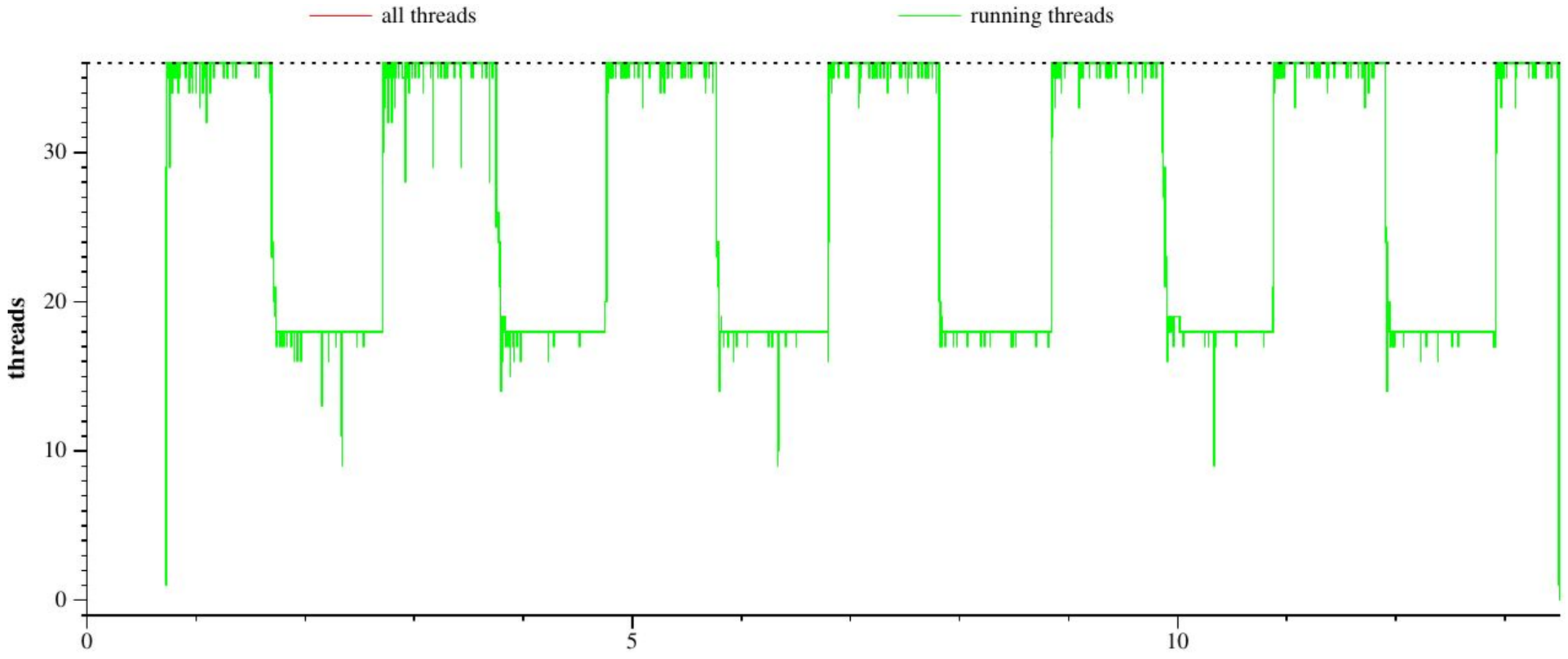
EEVDFS-host: Dynamic adaptation of DoP in the guest



EEVDFS-host, EEVDFS-guest

- OMP_WAIT_POLICY=*active*: 19.95 ± 0.5 seconds
- OMP_WAIT_POLICY=passive: 22.43 ± 0.09 seconds
- OMP_WAIT_POLICY=active + Dynamic adaptation of DoP: 13.02 ± 0.23 seconds (1.53x)

scx_central-host: Dynamic adaptation of DoP in the guest



scx_central-host, EEVDFS-guest

- OMP_WAIT_POLICY=*active*: 19.79 ± 0.6 seconds
- OMP_WAIT_POLICY=passive: 32.65 ± 0.08 seconds
- OMP_WAIT_POLICY=active + Dynamic adaptation of DoP: 12.71 ± 0.05 seconds (1.56x)

Question: What is the right way to make this policy available for sched_ext schedulers?

1. Provide the policy as a patch for the sched_ext kernel
2. Provide the policy as a set of eBPF programs
3. **Provide an example scx scheduler that includes the policy**

Thoughts?

Requirements for the example scx scheduler

- It needs to know the vcpu_id associated with the task_struct of the vCPU threads
- It needs to set-up and access the shared memory between the host and the guest schedulers
- It needs to detect precisely when a vCPU becomes a phantom, and when a pCPU becomes idle
  i.e. precise timestamps for context-switches and wake-ups
- It needs to process the history of context-switches and wake-ups involving phantom vCPUs and idle pCPUs at the end of every scheduler tick

How much of it is feasible?

Contact:
himadrics@pm.me