Methods for Emulation of Multi-Core CPU Performance

Tomasz Buchert[1] Lucas Nussbaum[2] Jens Gustedt[1]

[1] INRIA Nancy – Grand Est

[2] LORIA / Nancy - Université

Validation of distributed systems

Approaches:

- Theoretical approach (paper and pencil)
  - ☺ the most general results and understanding
  - ☹ very hard (leads to unsolvability results)
- Experimentation (real application on a real environment)
  - ☺ realistic context, credibility
  - ☹ difficulty of preparation and control, questionable reproducibility
- Simulation (modeled application inside modeled environment)
  - ☺ very simple and perfectly reproducible
  - ☹ experimental bias, possibly unrealistic
- Emulation (real application inside a modeled environment)
  - ☺ control over the experiment parameters
  - ☹ difficult

Emulation

The perfect emulated environment should emulate (independently):

- Network bandwidth, latency, topology
- Memory capabilities
- Background noise (network, faults)
- CPU speed and its features

Some parts implemented in **Wrekavoc** – a tool to define and control heterogeneity of the cluster

In this talk, however, we specifically concentrate on

Emulation

The perfect emulated environment should emulate (independently):

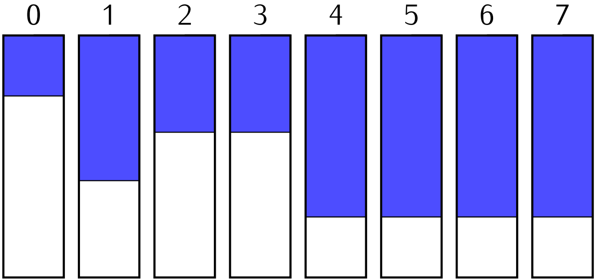
- Network bandwidth, latency, topology
- Memory capabilities
- Background noise (network, faults)
- CPU speed and its features

Some parts implemented in **Wrekavoc** – a tool to define and control heterogeneity of the cluster

In this talk, however, we specifically concentrate on

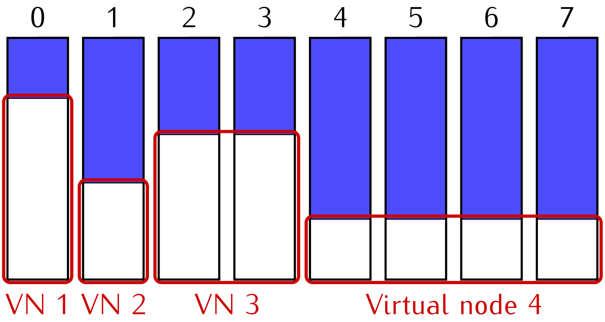
Emulation of CPU speed

Our goal



(1) control over the speed of each CPU/core independently

Our goal



(2) ability to create separately scheduled zones of tasks

Existing methods (CPU-Freq)

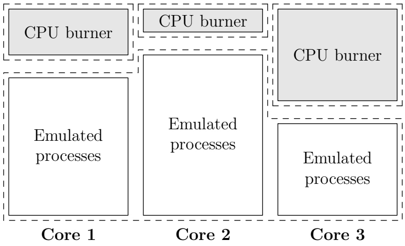
- Hardware solution to reduce heat, noise and power usage
- For:
  - no overhead of emulation
  - completely unintrusive
  - meaningful CPU time measure
- Against:
  - only a finite set of different frequency levels

Existing methods (CPU-Lim)

- Method available in Wrekavoc tool
- Algorithm:
  - if CPU usage $\geq$ threshold $\rightarrow$ send SIGSTOP to the process
  - if CPU usage $<$ threshold $\rightarrow$ send SIGCONT to the process
- $CPU\ usage = \frac{CPU\ time\ of\ the\ process}{process\ lifetime}$
- For:
  - easy and almost POSIX-compliant
- Against:
  - intrusive and unscalable
  - decision based on one process instead of global CPU usage
  - sleeping is indistinguishable from preemption

Existing methods (Fracas)

- Based on idea from KRASH (a load injection tool)
- Uses Linux Cgroups and Completely Fair Scheduler
- A predefined portion of the CPU is given to tasks burning CPU
- All other processes are given the remaining CPU time



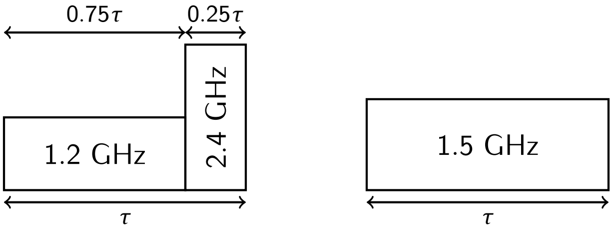
Existing methods (Fracas)

- Based on idea from KRASH (a load injection tool)
- Uses Linux Cgroups and Completely Fair Scheduler
- A predefined portion of the CPU is given to tasks burning CPU
- All other processes are given the remaining CPU time

- For:
  - unintrusive
  - scalable
- Against:
  - unportable to other systems
  - sensitive to the configuration of the scheduler

New methods (CPU-Gov)

- Generalization of CPU-Freq
- Alternates between two neighbouring hardware frequencies



New methods (CPU-Gov)

- Generalization of CPU-Freq
- Alternates between two neighbouring hardware frequencies

- For:
  - no overhead, unintrusive and meaningful CPU time measure (inherited from CPU-Freq)
  - continuous range of emulated frequency
- Against:
  - dependency on the hardware implementation (inherited from CPU-Freq)
  - special algorithm for small values of emulated frequency

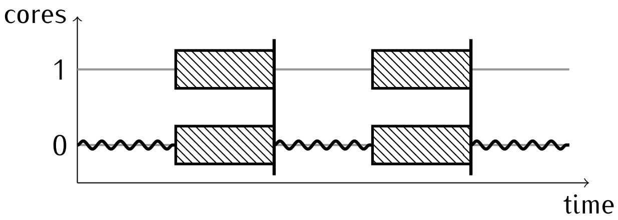
New methods (CPU-Hogs)

- Generalization of *CPU-burning* technique
- For each core there is a high-priority thread created
- They "burn" a required number of CPU cycles

- For:
  - simple and portable (POSIX)
  - does not rely on the hardware
- Against:
  - theoretical problems with scalability (not observed)

New methods (CPU-Hogs)

- Generalization of *CPU-burning* technique
- For each core there is a high-priority thread created
- They "burn" a required number of CPU cycles



Evaluation

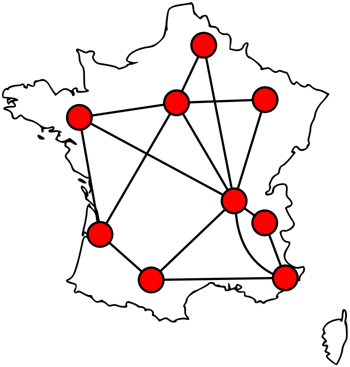
Microbenchmarks with different types of work:

- CPU intensive – running a tight computational loop
- IO bound – sending UDP packets over a network
- CPU and IO intensive – sleeping mixed with a computation
- multiprocessing – running multiple processes with CPU work
- multithreading – running multiple threads with CPU work
- memory speed (STREAM benchmark) – sustainable memory bandwidth

Evaluation (cont.)

- Tested with 1, 2, 4 and 8 emulated cores
- X-axis – emulated frequency
- Y-axis – speed perceived by the benchmark
- each test repeated 40 times, results = average with 95% confidence interval
- Evaluation performed on Grid'5000 platform

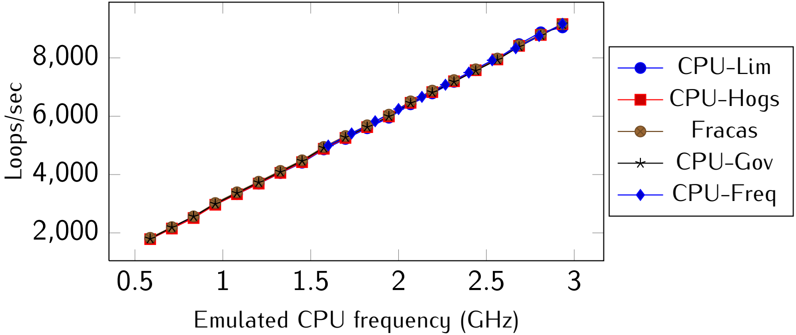
Grid'5000



- 9 sites, 1528 machines

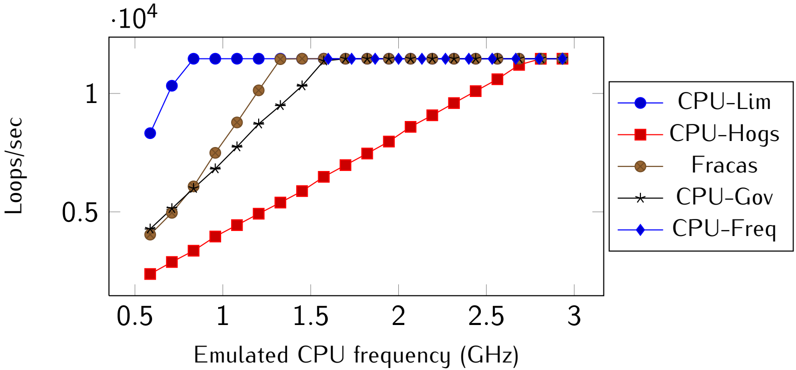
  Lille, Rennes, Orsay, Nancy, Bordeaux, Lyon, Grenoble, Toulouse, Sophia
- Dedicated to research on distributed systems and HPC

CPU intensive work (one core)



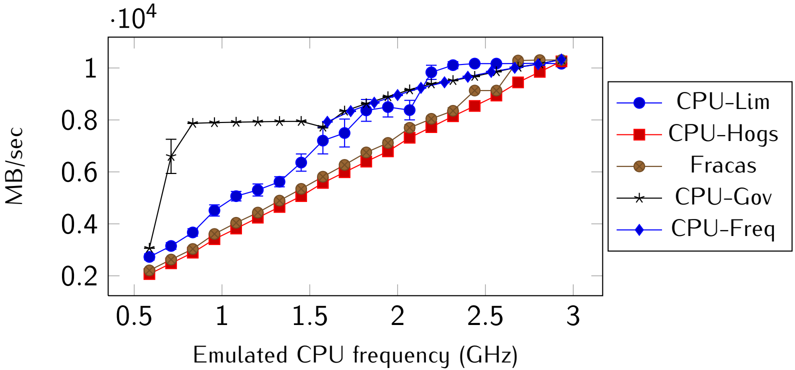
All methods work as expected.

IO-intensive work (one core)



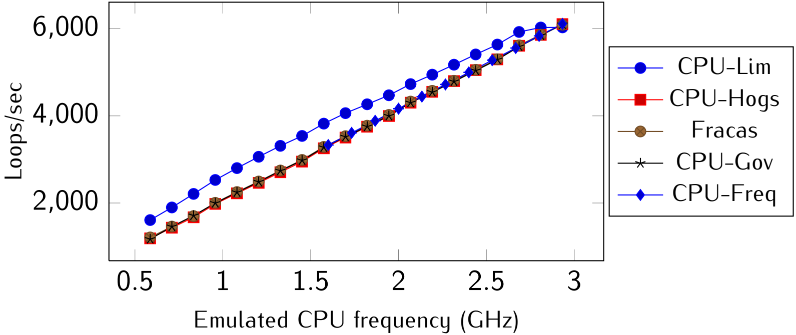
IO operations should not scale with CPU frequency.

Memory speed (one core)



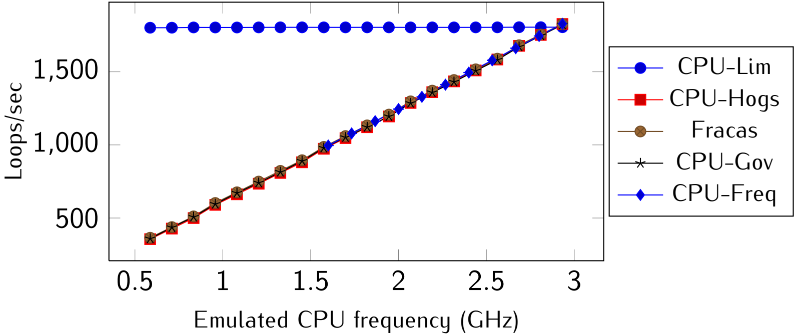
Ideally, memory speed would not be scaled as well.

Computing and sleeping workload (one core)



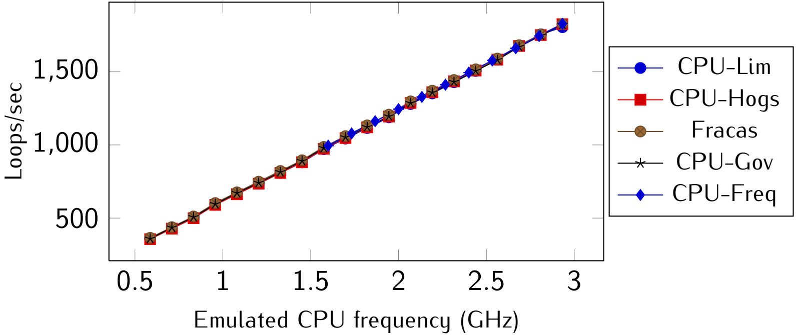
The relation should be proportional, but CPU-Lim's is not.

Multiprocessing benchmark (one core)



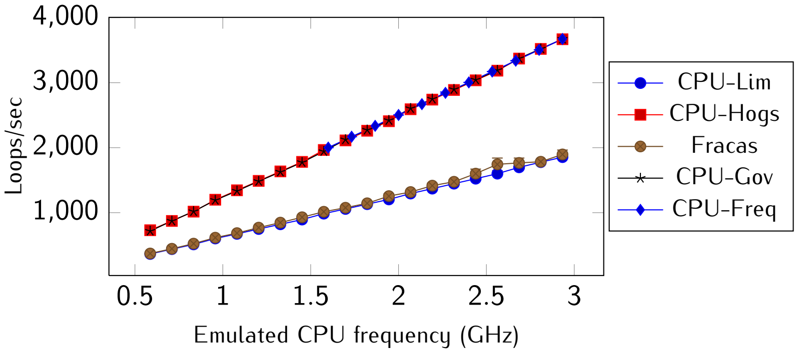
This relation should be proportional again (but CPU-Lim's is not).

Multithreading benchmark (one core)



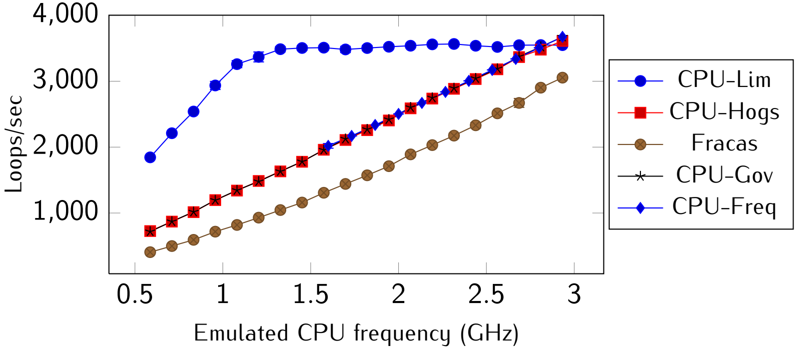
The execution speed scales with the frequency.

Multithreading benchmark (two cores)



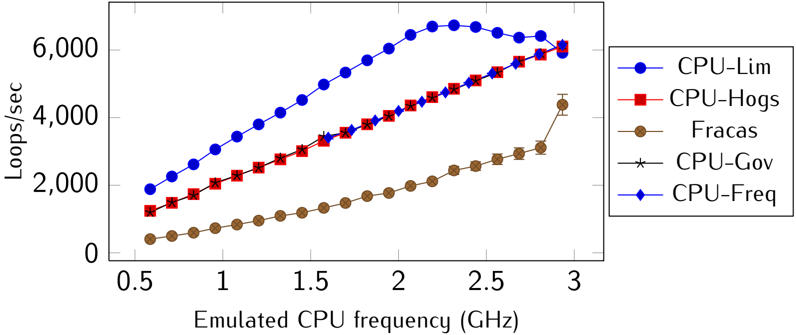
CPU-Lim and Fracas run twice as slow as CPU-Freq.

Multiprocessing benchmark (two cores)



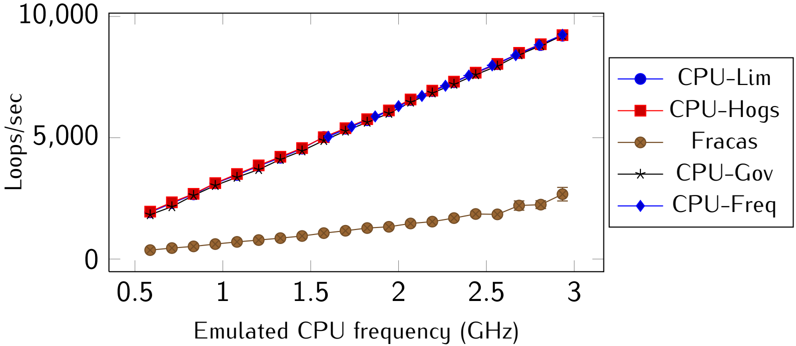
CPU-Lim and Fracas fail in this benchmark too.

Multiprocessing benchmark (four cores)



What happened here?

Multiprocessing benchmark (eight cores)



Fracas fails once more (but CPU-Lim doesn't!).

Summary of the evaluation

- CPU-Freq:
  - very good results
  - coarse granularity
- CPU-Lim:
  - not scalable due to implementation, intrusive
  - higher variance
  - controls processes, not threads
- Fracas:
  - good behavior for a single-task workload
  - scalable
  - bad behavior for multitask workload
  - behavior differs from one version of Linux to another

Summary of the evaluation (cont.)

- CPU-Gov and CPU-Hogs:
  - improvement over previous methods
  - good and stable behavior in virtually every benchmark
  - scalability
  - independent from the underlying OS

Future work

- Emulate memory bandwidth
- Emulate other aspects of CPU
- Test the methods with real-life applications
- Integrate the best methods into an open source, user-friendly emulator (Wrekavoc)

Conclusions

- Presented CPU-Hogs, CPU-Gov and previously existing methods
- Compared them by running a set of microbenchmarks
- Evaluated experimentally on Grid'5000
- New methods show a big improvement in the quality of emulation

Thanks for listening.

Questions?