Accurate emulation of 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
- Performance and number of CPUs
- Memory capabilities
- Background noise (network, CPU, faults)

Already implemented in **Wrekavoc** – a tool to define and control heterogeneity of the cluster (but not perfect yet!)

In this talk, however, we specifically concentrate on

Emulation

The perfect emulated environment should emulate (independently):

- Network bandwidth, latency, topology
- Performance and number of CPUs
- Memory capabilities
- Background noise (network, CPU, faults)

Already implemented in **Wrekavoc** – a tool to define and control heterogeneity of the cluster (but not perfect yet!)

In this talk, however, we specifically concentrate on

Emulation of CPU

CPU emulation

Various elements of CPU architecture could be emulated:

- speed
- number of cores
- sizes and properties of caches (and topology thereof)
- memory access speed (especially for NUMA systems)

In this talk, we will talk about

CPU emulation

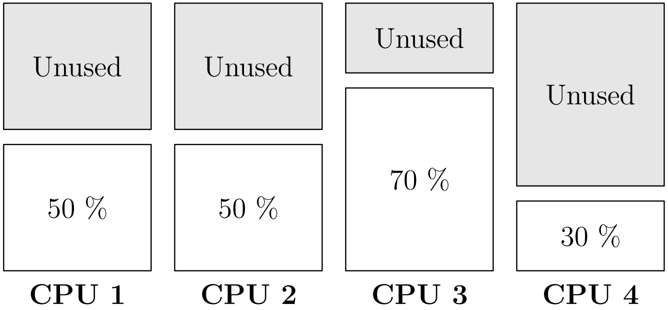
Various elements of CPU architecture could be emulated:

- speed
- number of cores
- sizes and properties of caches (and topology thereof)
- memory access speed (especially for NUMA systems)

In this talk, we will talk about

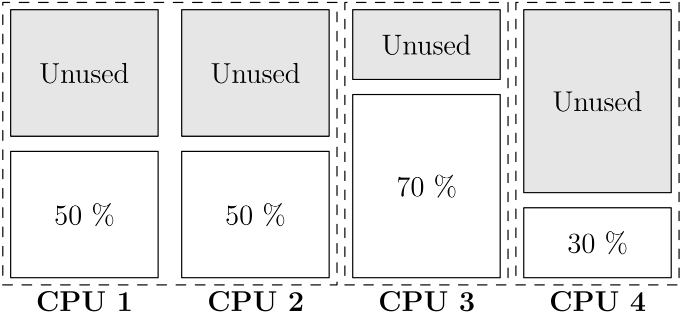
Degradation of CPU speed

An example



(1) controlling speed of each CPU/core independently

An example (continued)



(2) being able to create separated scheduling zones

Dynamic frequency scaling (CPU-Freq)

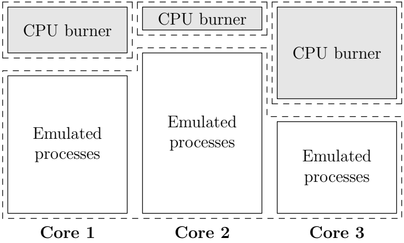
- AKA Intel Enhanced SpeedStep or AMD Cool'n'Quiet
- Hardware solution to reduce:
  - heat
  - noise
  - power usage
- For:
  - no overhead of emulation
  - completely unintrusive
  - meaningful CPU time measure
- Against:
  - only a finite set of different frequency levels

CPU-Lim

- Method available in Wrekavoc
- Algorithm:
  - if CPU usage ≥ threshold → send SIGSTOP to the process
  - if CPU usage < threshold → 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

Fracas

- Based on idea from KRASH (load injection tool) idea
- 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

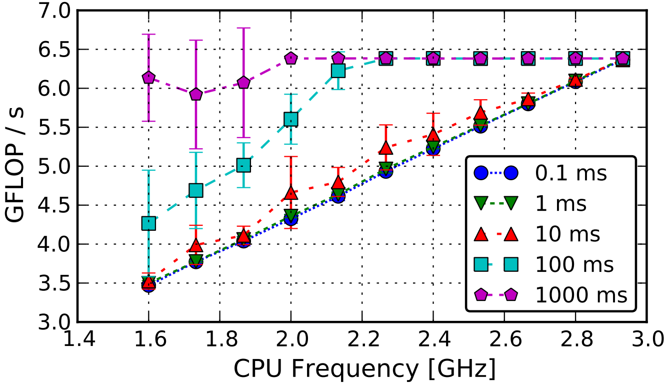


Fracas

- Based on idea from KRASH (load injection tool) idea
- 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

Fracas and latency of the scheduler

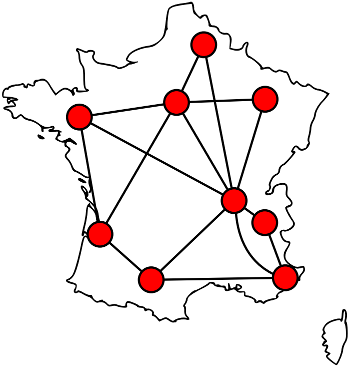


The smaller the latency, the better the emulation

Evaluation

- Based on different types of work:
  - CPU intensive (Linpack benchmark)
  - IO bound
  - multiprocessing
  - multithreading
  - memory speed (STREAM benchmark)
- X-axis – emulated frequency
- Y-axis – speed perceived by the benchmark
- each test repeated 10 times, results = average
  95% confidence interval using t-Student distribution
- Evaluation performed on Grid'5000 platform
  - nodes with two quad-core Intel Xeon X5570 processors
  - nodes with a pair of single-core AMD Opteron 252 processors

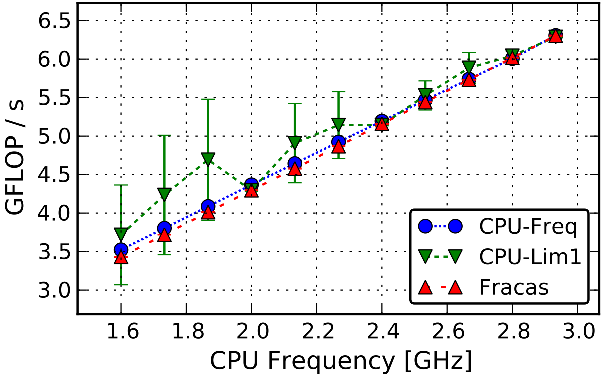
Grid'5000



- 9 sites, 1600 machines

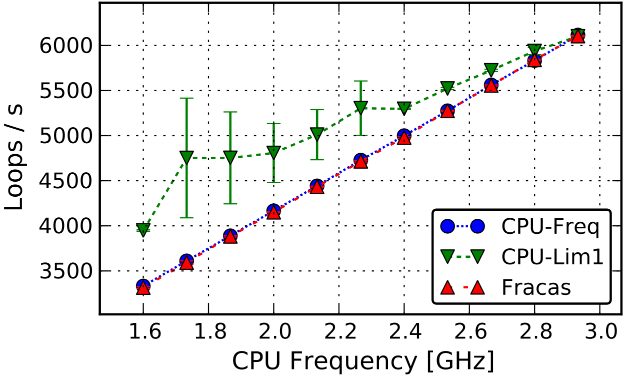
  Lille, Rennes, Orsay, Nancy, Bordeaux, Lyon, Grenoble, Toulouse, Sophia
- Dedicated to research on distributed systems and HPC

CPU intensive work



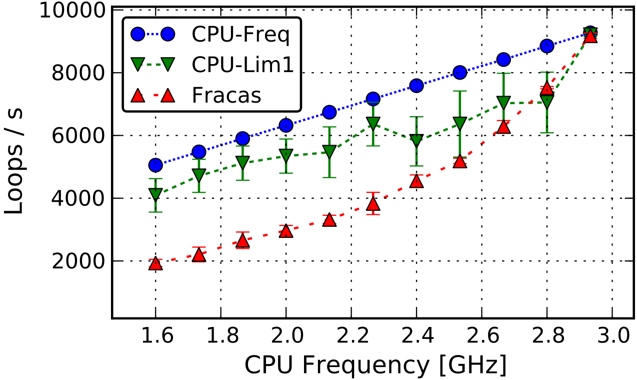
CPU-Lim is less predictable (the outcome has higher variance)

IO-bound work



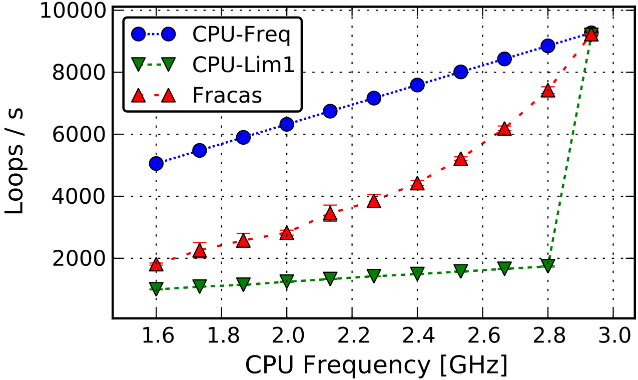
CPU-Lim gives (unfair) advantage to IO-bound tasks

Multiprocessing



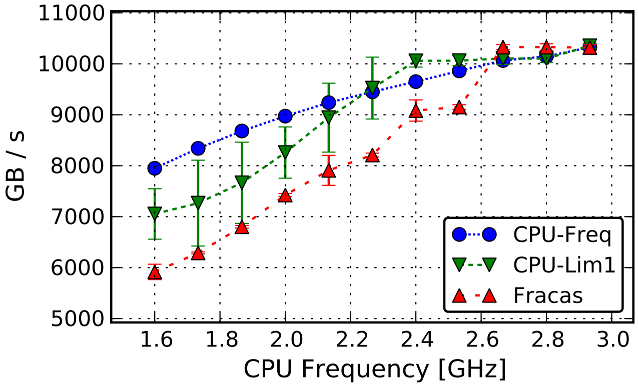
Fracas can't emulate CPU for multitask computation

Multithreading



CPU-Lim controls *processes* instead of scheduling entities

Memory speed



Memory speed is affected differently by each method

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

Future work

- Explore other approaches
- Improve Fracas to cover multitasking
- Emulate memory bandwidth
- Emulate other aspects of CPU
- Integrate Fracas into Wrekavoc
- Take over the world :)

Conclusions

- Presented Fracas, a method for CPU performance emulation based on Linux cgroups
- Compared with CPU-Freq and CPU-Lim (Wrekavoc)
- Evaluated experimentally on Grid'5000
- None of the methods is perfect:
  - CPU-Freq: coarse grained
  - CPU-Lim: implementation problems, not scalable
  - Fracas: works perfectly in single thread/process case, needs work in multithread/process case

Questions?