Problem Description

Induce a partial CPU load using existing applications. What is a controlled partial CPU load good for?
- Partial load experiments with reproducible results
- Relation between CPU load and application performance
- Relation between CPU load and power consumption
- Impact of a changing CPU load in a neighboring virtual machine
- OPTimal CPU pinning and frequency scaling configurations
- Effects and benefits of frequency scaling, TurboBoost and NUMA

CPU Load is the fraction of time CPUs spend executing threads.

Duty Cycle is the fraction of time an application can run.

Given a workload capable of hogging a CPU, how can we control the average CPU load and sustain a partial load?

By the number of workload instances:
- What if the workload runs a varying number of threads?
  - Only a limited number of partial load values and only one distribution of load across CPUs (“all or nothing”) can be achieved.
- Dark column (〈) in Figure 1 show that the approach works for mostly single-threaded applications (unpinned, unpinned), but fails otherwise.

By the duty cycle of one workload instance:
- What if there are not enough threads to saturate all CPUs?
  - Then the CPU load will be lower than the duty cycle. Light column (〈) in Figure 1 show that the approach works for heavily multi-threaded applications (unpinned, unpinned), but fails otherwise.

By the duty cycle of #CPUs workload instances:
- What if such workload instance runs multiple threads?
  - Multi-threadedness causes the load to exceed the duty cycle, unless we synchronize duty cycle enforcement across all instances. Yet we do not want any synchronisation among parallel workloads.

Figure 2 shows that the approach works for the single-threaded luindex (〈), tends towards higher loads for the slightly multi-threaded h2 (〈) and spins out of control for the heavily multi-threaded sunflow (〈). The experiment uses a synthetic one-threaded load trace and enforces a duty cycle equal to the target CPU load.

Figure 3 shows how Showstopper resolves the load control problem shown in Figure 2. Even #CPUs instances of the heavily multi-threaded sunflow can be forced to follow the required CPU load target.

Our Solution: Showstopper

Showstopper is a software tool that uses feedback control mechanisms to achieve and maintain a partial CPU load.

- We start #CPUs workload instances using Showstopper.
- Showstopper’s feedback controller responds to system load changes.
- The controller’s input is the CPU load observed on the system.
- The controller’s output is the duty cycle to enforce.
- The controller adjusts the duty cycle to meet the load target.

Figure 3 shows how Showstopper resolves the load control problem shown in Figure 2. Even #CPUs instances of the heavily multi-threaded sunflow can be forced to follow the required CPU load target.

Example Application

Let us define and replay a synthetic load trace, using Showstopper to start and control 12 instances of luindex.

Figure 4 shows the first load definition in a format readable by Showstopper (left). Next Showstopper starts 12 instances of a workload and follows the load trace (right). (The ‘X’ symbol is replaced by a workload instance number.)

Power consumption, application performance...

We use the “staircase” load trace to avoid a separate warm-up period for each partial load experiment. We let the workloads warm up once and use Showstopper to follow a staircase: 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960, 81920.

Two VMs with 16 CPUs share a physical server with 32 CPUs. The two VMs do not compete for CPU time. We intend to investigate other forms of performance interference. One VM runs 16 uncontrolled instances of h2. The other VM runs 16 instances of luindex controlled by Showstopper according to the “staircase” trace.

We would like to investigate:
- How application throughput of luindex depends on the CPU load.
- How application throughput of the uncontrolled h2 in one VM depends on the CPU load induced in the other VM.
- Whether it is better to pin the two VMs to separate NUMA nodes or to let them migrate freely.
- Whether the performance of the VM running h2 in absence of pinning (●) is inferior to the pinned performance (●) whenever the CPU load in the VM running luindex is lower than 80%.
- How CPU pinning of VMs and CPU load influence system power consumption. Figure 8 shows that the unpinned configuration (●) yields a higher power consumption than the pinned one (●) when the VM running luindex is lightly loaded.

Figure 6 shows observed CPU load (%) and throughput (●) of luindex when following the “staircase” trace using Showstopper. The light grey band represents the target load.

Figure 7 shows the throughput of uncontrolled h2 running at the same time as luindex in Figure 6 on a neighboring VM when the two VMs are pinned to separate NUMA nodes (●) or unpinned (●).

Figure 8 shows power consumption whilst running uncontrolled h2 and controlled luindex (●) as shown in Figures 6 and 7. The two VMs are pinned to separate NUMA nodes (●) or unpinned (●).