Do nothing fast:
How to scale idle CPUs ?

Mathieu Desnoyers
EfficiOS Inc.

How it All Started...

- RSEQ mm_cid feature added a raw spinlock in the scheduler context switch fast-path,
- This caused regressions on some workloads at scale (reported by Intel, AMD and Oracle),
- The fix ***REMOVING*** the spin lock fixed those regressions, but it introduced performance regressions on the hackbench workload at scale (reported by AMD),
- I could reproduce on a 2-sockets, 192-core, 384 HW threads EPYC 9654.
- Hackbench workload: thread mode, 32 groups, 20 fds per group, pipes, 100 bytes messages, 480k loops.

Several Approaches Attempted

- Extend cpu idle state for 1ms
- Favor almost idle previously used CPUs for wake affine
- Skip queued wakeups only when L2 is shared
- Rate limit task migration
- Bias runqueue selection towards almost idle prev CPU
- Bias runqueue select towards prev CPU
  - Introduce UTIL_FITS_CAPACITY feature
  - Introduce SELECT_BIAS_PREV to reduce migrations

Effects of Those Approaches

- Speed up hackbench workload from 49s to 29s-35s.
- Reduce the migration rate.

Client-server Workload Regression

- Bias towards prev runqueue helps hackbench (N:M), but regresses client-server workloads (1:1).

Issue with Work Conserving

- RT definition: In a system with N CPUs, the N "higest priority" tasks must be running (in real-time),
- Being work conserving, the Linux fair scheduler never leaves cores idle when there is work to do,
- On large systems, this correlates with observation of high migration rates for certain workloads and CPU utilization,
- It is a concern for:
  - Runqueue locking overhead/contention,
  - Cache and NUMA misses of workload memory accesses.

EfficiOS

Task Placement (WIP)

- Communication patterns based on wakeups (waker/wakee),
- Convergence of task placement based on a communication pattern predictor (task packing),
- Predictor is implemented with per-task counters based on the hardware hierarchy (numa nodes, clusters (LLC), cores),
- Requires a new runqueue metric.

Scheduler Runqueue Metrics

- Average (includes sleeping/blocking tasks):
  - Utilization,
  - Runnable,
  - Load,
- Enqueued estimate:
  - Utilization,
  - ***Runnable (new metric)***

Runqueue Selection Algorithm

- Switch with waker (WF_SYNC),
- Find NUMA node, cluster, core with maximum connectivity for task,
- Perform backtracking to find core with enough capacity, while keeping track of core with minimal overcommit.
  - If target core can fit task without overcommit, use it,
  - Search across cluster (LLC) for a core with enough remaining capacity for task,
  - Search across NUMA node for a core with enough remaining capacity for task,
  - Search entire machine for a core with enough remaining capacity for task,
  - If all fails, use the core with minimal overcommit.

EfficiOS

Current Results

- Hackbench performance on par with mainline fair scheduler (~49s),
- On a 2 NUMA nodes machine, the packing works a little too well and leaves all CPUs of node 1 at 66% idle, compared to CPUs of node 0 which are 10% idle.
- In comparison, a mainline kernel is 40% idle on all CPUs.
- What has not been done so far:
  - Integration with task balancing algorithm,
  - Integration with NUMA balancing.