Generic Concurrency Restriction

Alex Kogan
Oracle Labs

joint work with Dave Dice
The Scalability Collapse problem

Growing #threads circulating through a saturated lock causes the overall application performance to fade
Why does it happen?

• Competition over shared resources
  – computing cores, pipeline availability
  – last-level caches, DRAM channel and interconnect bandwidth
  – thermal budget
  – etc.
Typical (spin-then-park) lock
Introducing GCR
What is GCR?

• The goal: keep the lock saturated by as few threads as possible

• Lock-agnostic wrapper
  – intercepts lock acquisitions calls
  – decides which threads would be allowed to proceed (aka remain active)
  – other threads form a queue and block (aka become passive)
  – periodically shuffle between active and passive
    • avoid starvation, achieve long-term fairness
GCR-NUMA

• Maintains the set of active threads in a “NUMA-aware” way
  – composed of threads running on the same socket

• Can convert any lock into a NUMA-aware one
Preliminary results

AVL tree microbenchmark
2 Intel Xeon E5-2630 v4 procs, 40 logical CPUs in total

Throughput (op/ns)

# threads

MCS (spin)
MCS (spin) + GCR
MCS (spin) + GCR NUMA
Malthusian (spin)
Preliminary results

AVL tree microbenchmark
2 Intel Xeon E5-2630 v4 procs, 40 logical CPUs in total

Throughput (op/ns) vs. # threads
Preliminary results

AVL tree microbenchmark
2 Intel Xeon E5-2630 v4 procs, 40 logical CPUs in total
Wrap up

• GCR avoids the scalability collapse
  – keeps the lock saturated with a few threads
  – passivates all the rest
  – reduces contention and consumption of valuable resources

• Future work: adaptivity
  • disable GCR under no/light contention