Scheduler Soft Affinity: Dynamic Workload Partitioning

Subhra Mazumdar
Oracle Linux

Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

CPU Resource Management

- Multiple instances of workload are consolidated in one system
  - Multiple VMs or Containers
  - Autonomous Database
    - Oracle Multitenant
    - Multiple Container Database/Pluggable Database instances

CPU Resource Management

- Instances can be hard partitioned
  - *sched_setaffinity(2)* or *cpusets* to e.g. a NUMA node
  - One instance can't use the CPUs of another
- Instances can be free inside the system
  - Can be scheduled on any CPU (no affinity)
  - *cpu.shares* for fair share
  - Cache interference and coherence

Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

Dynamic Partitioning

- Goal: Best of both worlds
  - Allow bursting out of partition
  - Minimize cache interference and coherence when busy
  - Have negligible overhead

AutoNuma Balancer?

- Can work for NUMA partitions (LLC)
- High reaction time
  - Periodic scanning
- Ineffective if memory is spread across NUMA nodes

AutoNuma Balancer?

- Motivational experiment
  - *numactl* used to bind memory (2 DB instances, 2 socket system)
  - AutoNuma Balancer not migrating threads
  - AutoNuma Balancer ON vs OFF did not make a difference

Dynamic Partitioning

- ”Soft” partition of *any* set of CPUs
  - LLC (NUMA) can be most effective on x86 Intel systems
  - L2/L1 can be beneficial for other platforms
  - APIs

CPU shares

- Soft Affinity is orthogonal to CPU shares
  - CPU shares decides *how many* cycles to consume
  - Soft affinity decides *where* to preferably consume those cycles
  - Can be used together

Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

Implementation

- Define a preferred set of CPUs
  - *cpu_preferred* in addition to *cpu_allowed*
- Use *cpu_preferred* for *want_affine*
- LLC search:

```
if (cpu_preferred == cpu_allowed)
        search cpu_allowed;
else
        search cpu_preferred;
        search (cpu_allowed – cpu_preferred);
```

Implementation

- Only changes the *push* side of the scheduler
  - Idle balancing unchanged
  - *Implicitly soft*
- Only implemented for CFS class
  - Child inheritance

Implementation

API

- New system call to specify soft partition
  - sched_setaffinity2
  - Extra parameter for affinity type
  - HARD_AFFINITY == sched_setaffinity
  - SOFT_AFFINITY == ”soft” affinity
- Other option: extending cpuset

Implementation

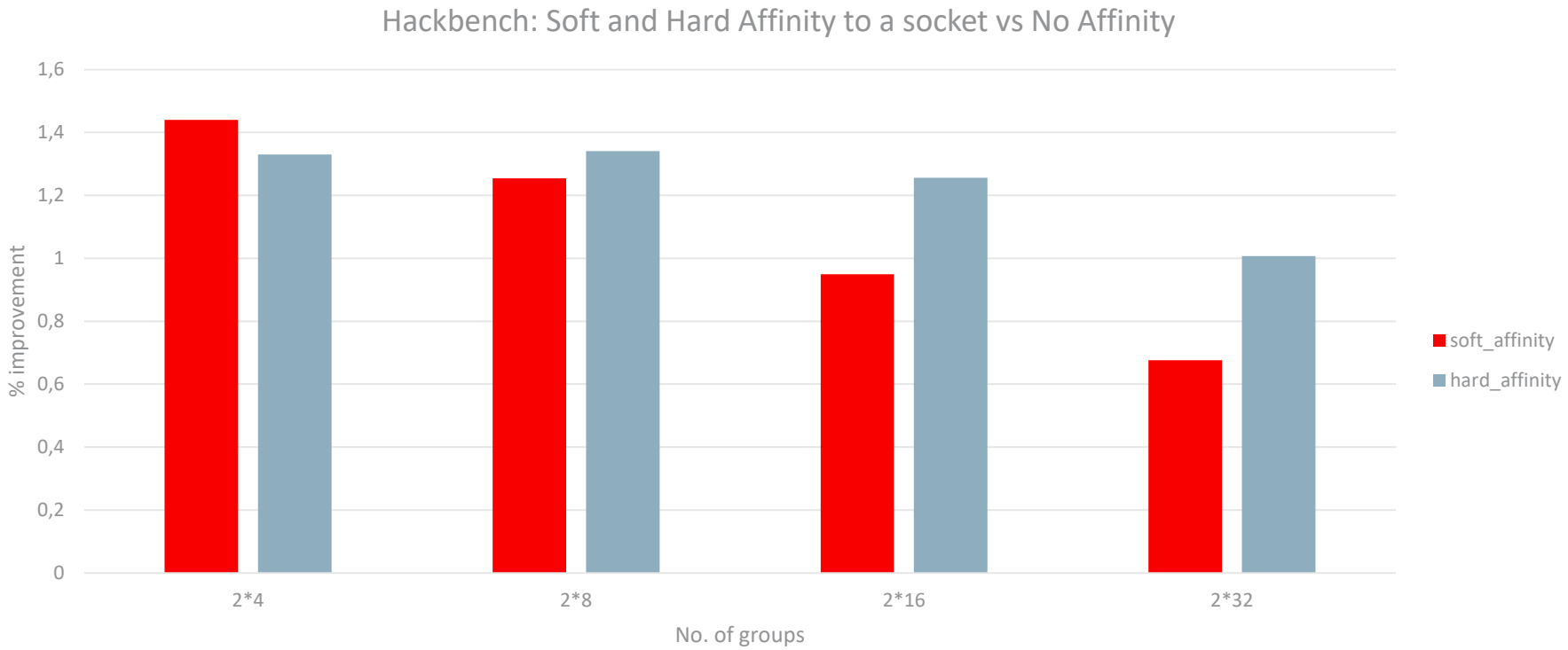
Example snippet

```
...
        cpu_set_t set;
        pid_t pid;
        CPU_ZERO(&set);
        for (i=0; i<=3; i++)
                CPU_SET(i, &set);
        pid = getpid();
        rc = sched_setaffinity2(pid, sizeof(cpu_set_t), &set, SOFT_AFFINITY);
...
```

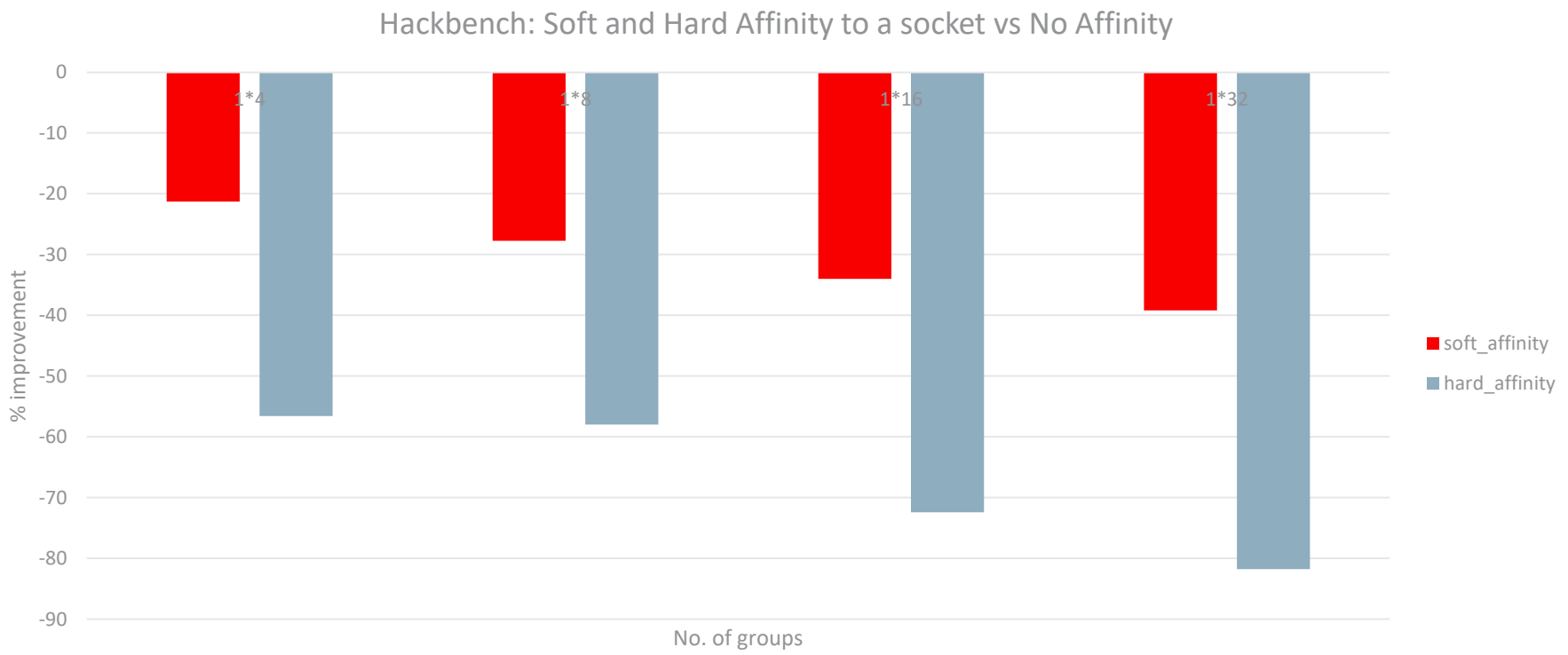
Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

Hackbench: 2 instances, 2 socket NUMA system, 22 cores per socket



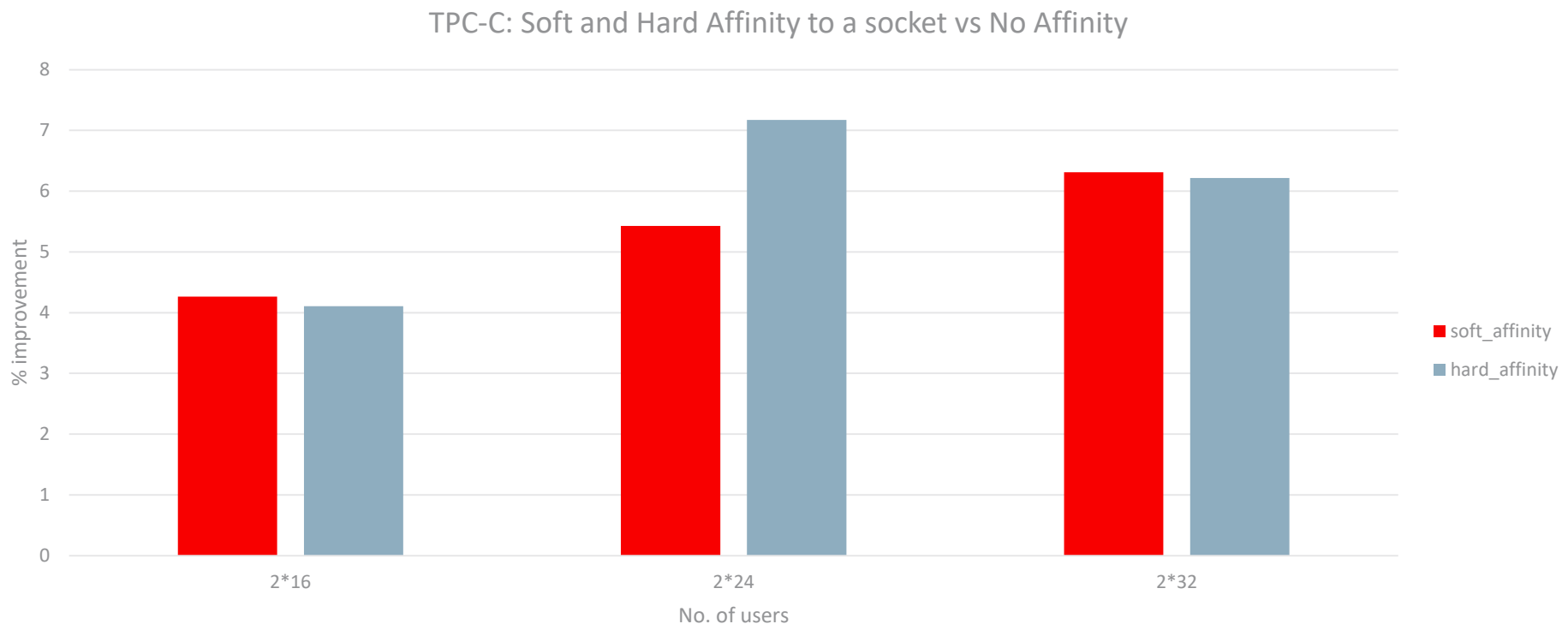
Hackbench: 1 instance, 2 socket NUMA system, 22 cores per socket



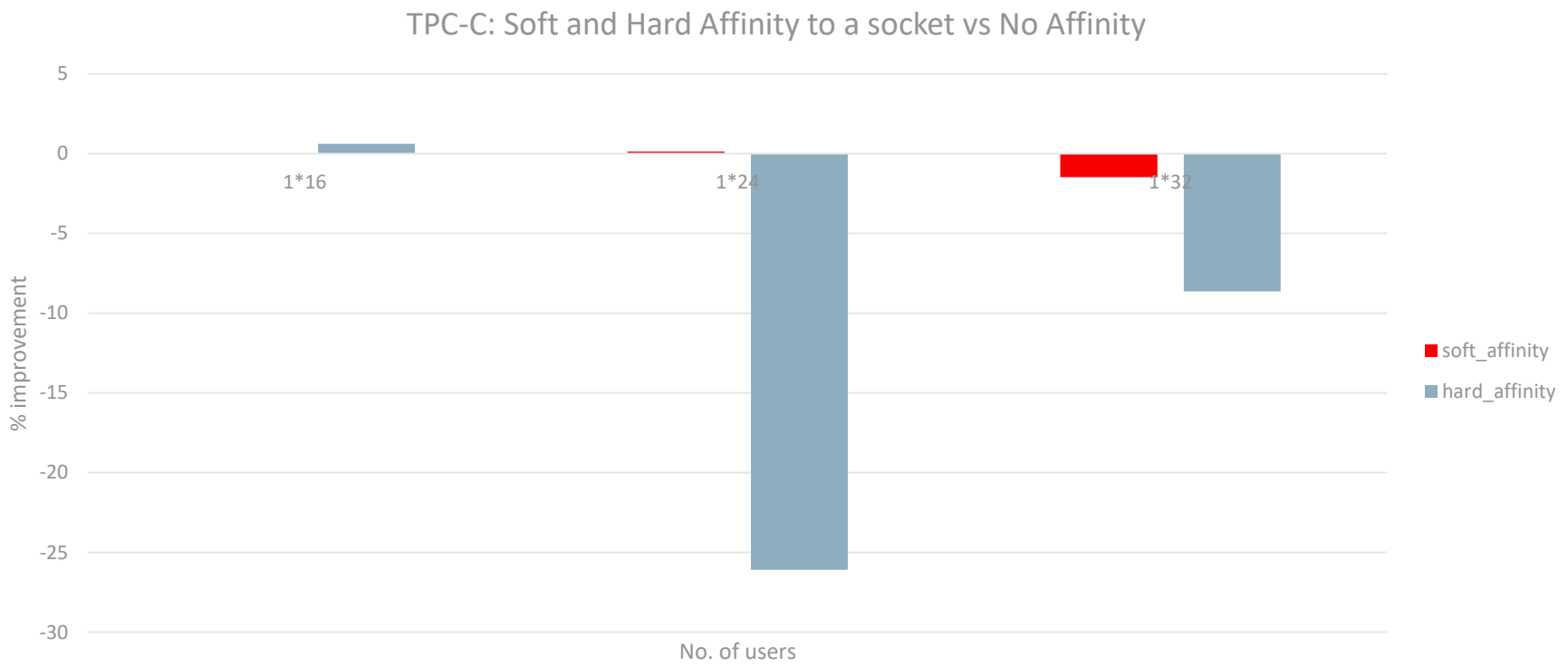
Results

- Small improvement for Hackbench 2 instances
  - some benefit of islolating LLC
  - Little room for improvement
- Big regressions for Hackbench 1 instance
  - Not using all CPUs efficiently
  - Soft Affinity better than Hard Affinity

TPC-C: 2 instances, 2 socket NUMA system, 22 cores per socket



TPC-C: 1 instance, 2 socket NUMA system, 22 cores per socket

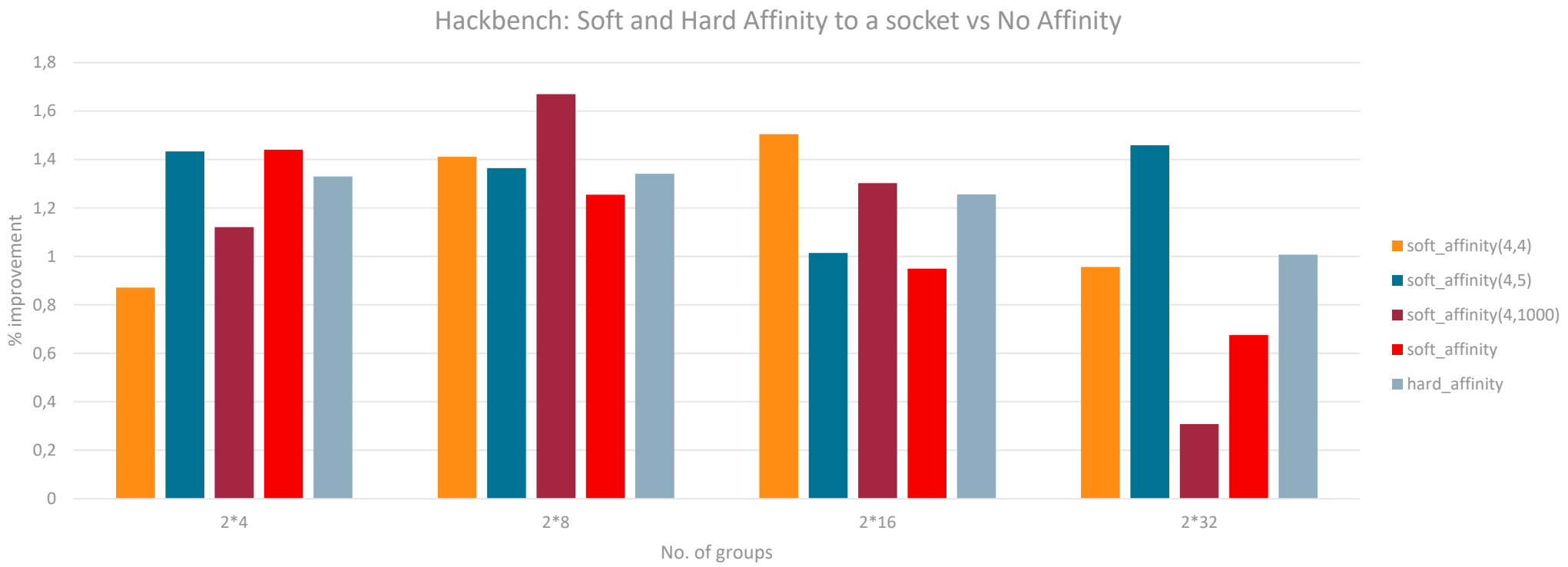


Load Based Soft Affinity

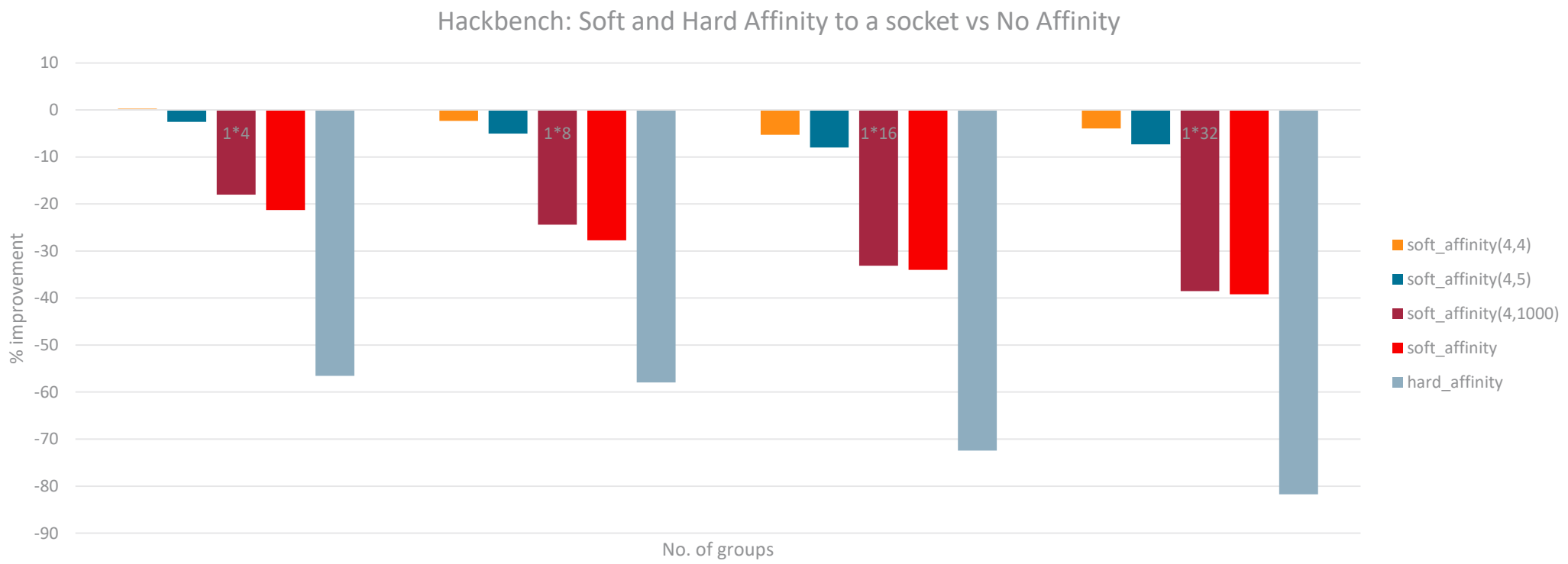
- Conditionally use *cpu_preferred* for *want_affine*
  - Compare two CPUs from *cpu_preferred* and (*cpu_allowed – cpu_preferred)* in O(1)
  - Scheduler tunables: *sched_preferred* and *sched_allowed*

```
cpu_x = cpumask_any(cpu_preferred);
cpu_y = cpumask_any(cpu_allowed – cpu_preferred);
If (sched_preferred * cpu_x utilization > sched_allowed * cpu_y utilization)
        want_affine = 0;
else
        want_affine = 1;
```

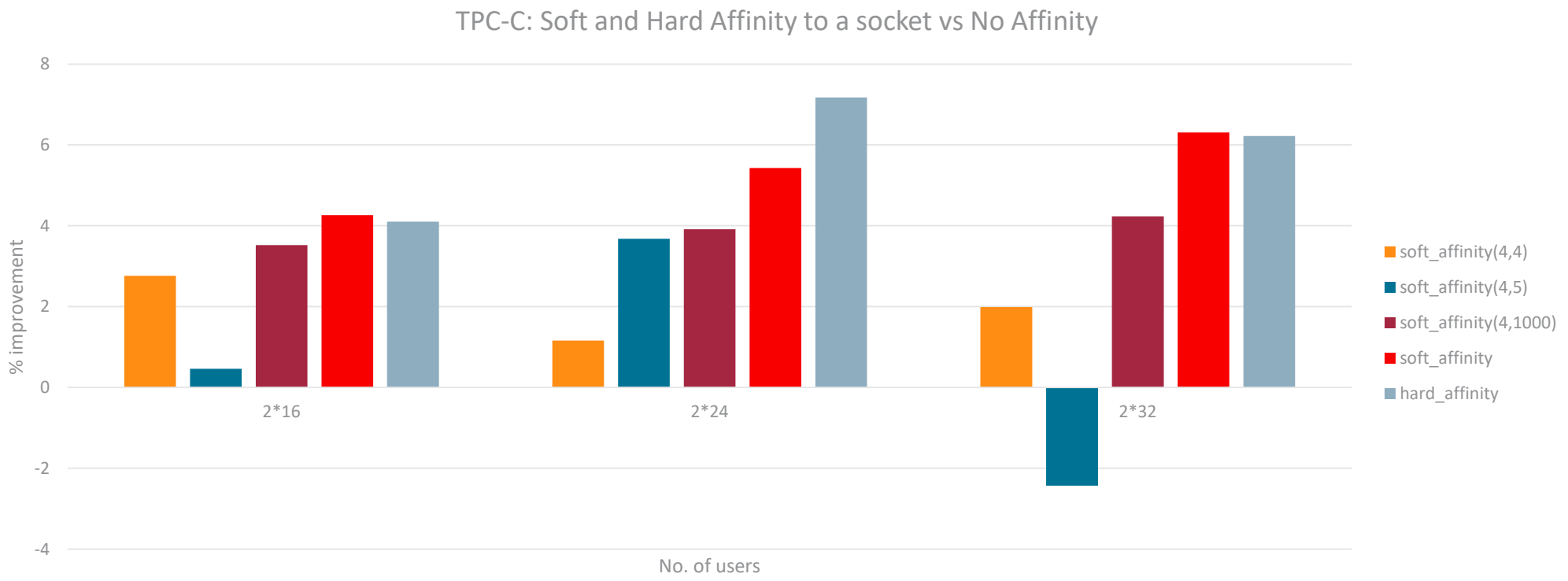
Hackbench: 2 instances, 2 socket NUMA system, 22 cores per socket



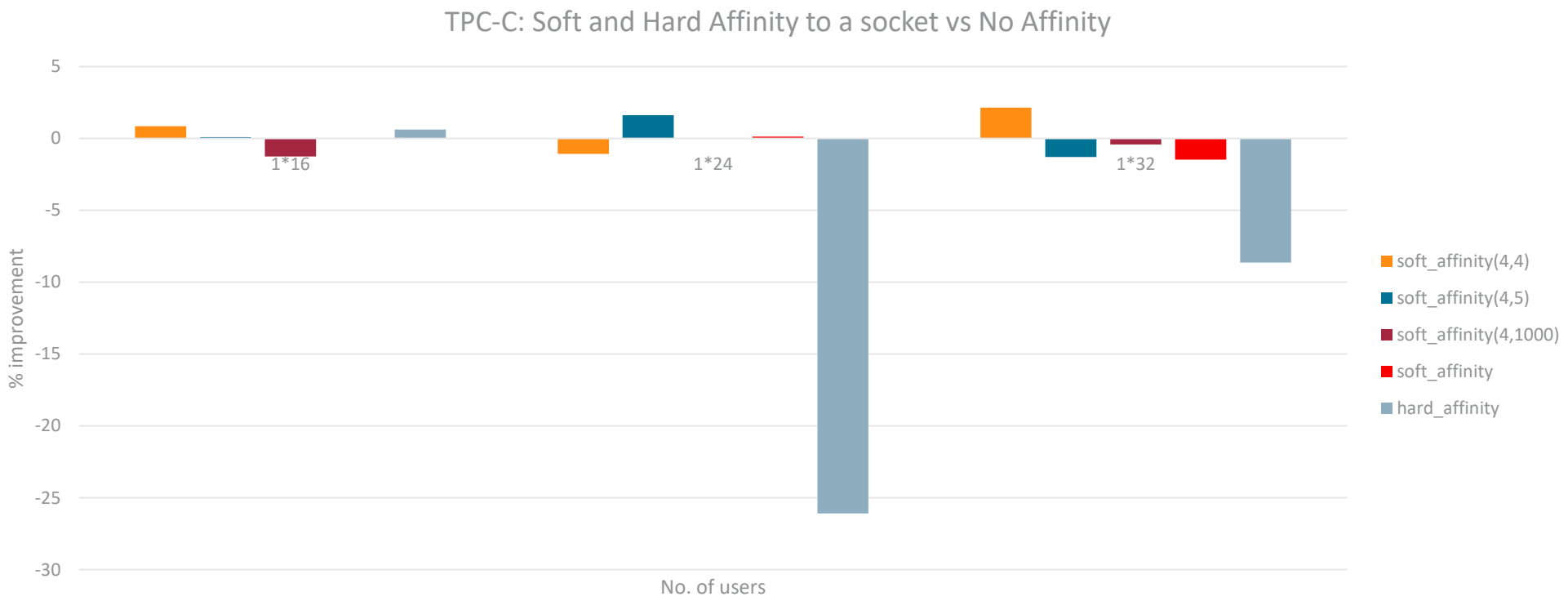
Hackbench: 1 instance, 2 socket NUMA system, 22 cores per socket



TPC-C: 2 instances, 2 socket NUMA system, 22 cores per socket



TPC-C: 1 instance, 2 socket NUMA system, 22 cores per socket



No overhead when Soft Affinity *not* used

Hackbench	No_Affinity % change	Hard_Affinity % change
2x4	0.10590459	0.30898275
2x8	0.128548	0.55474365
2x16	0.61016786	0.90104716
2x32	0.86854058	1.01481597
1x4	0.47585608	0.43398421
1x8	0.45468479	0.33351682
1x16	0.61327776	0.64039494
1x32	0.11759902	0.62852034

Results

- *numactl* used for each DB instance
- For TPC-C load based Soft Affinity (4,1000) works best
  - *harder* Soft Affinity similar to basic implementation
  - Improvements are more (~4% for 2 DB instance) than regressions (~-0.5% 1 DB instance)
- Other experiments
  - Spreading memory across NUMA nodes has *similar* results
  - Sub-NUMA partitions (soft or hard) had no benefit

Outline

1. CPU Resource Management
2. Dynamic Partitioning
3. Implementation
4. Results
5. Future Work

Future Work

- Testing on bigger (4 and 8 socket) systems, more DB instances
  - Find right tunable values
- Get rid of tunables?
  - Heterogeneous workload consolidation: per process tunables
  - Kernel does not have enough information of workload
    - cache sharing between threads
    - working set size
    - producer-consumer pattern

Questions?


ORACLE®