Automatic NUMA Balancing

Rik van Riel, Principal Software Engineer, Red Hat
Vinod Chegu, Master Technologist, HP
Automatic NUMA Balancing Agenda

• What is NUMA, anyway?
• Automatic NUMA balancing internals
• Automatic NUMA balancing performance
  • What workloads benefit from manual NUMA tuning
• NUMA tools
• Future developments
• Conclusions
Introduction to NUMA

What is NUMA, anyway?
What is NUMA, anyway?

• Non Uniform Memory Access
• Multiple physical CPUs in a system
• Each CPU has memory attached to it
  • Local memory, fast
• Each CPU can access other CPU's memory, too
  • Remote memory, slower
NUMA terminology

• Node
  • A physical CPU and attached memory
  • Could be multiple CPUs (with off-chip memory controller)

• Interconnect
  • Bus connecting the various nodes together
  • Generally faster than memory bandwidth of a single node
  • Can get overwhelmed by traffic from many nodes
HP Proliant DL580 Gen8 – NUMA topology

4-socket Ivy Bridge EX processor

# numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 262040 MB
node 0 free: 249261 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 262144 MB
node 1 free: 252060 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 262144 MB
node 2 free: 250441 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 262144 MB
node 3 free: 250080 MB
node distances:
node 0  1   2    3
0:   10  21  21  21
1:   21  10  21  21
2:   21  21  10  21
3:   21  21  21  10
HP Proliant DL980 G7 – NUMA topology
8-socket Westmere EX processor

# numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
node 0 size: 262133 MB
node 0 free: 250463 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19
node 1 size: 262144 MB
node 1 free: 256316 MB
node 2 cpus: 20 21 22 23 24 25 26 27 28 29
node 2 size: 262144 MB
node 2 free: 256439 MB
node 3 cpus: 30 31 32 33 34 35 36 37 38 39
node 3 size: 262144 MB
node 3 free: 255403 MB
node 4 cpus: 40 41 42 43 44 45 46 47 48 49
node 4 size: 262144 MB
node 4 free: 256546 MB
node 5 cpus: 50 51 52 53 54 55 56 57 58 59
node 5 size: 262144 MB
node 5 free: 256036 MB
node 6 cpus: 60 61 62 63 64 65 66 67 68 69
node 6 size: 262144 MB
node 6 free: 256468 MB
node 7 cpus: 70 71 72 73 74 75 76 77 78 79
node 7 size: 262144 MB
node 7 free: 255232 MB
node distances:
node 0 1 2 3 4 5 6 7
0: 10 12 17 17 19 19 19 19
1: 12 10 17 17 19 19 19 19
2: 17 17 10 12 19 19 19 19
3: 17 17 12 10 19 19 19 19
4: 19 19 19 19 10 12 17 17
5: 19 19 19 19 12 10 17 17
6: 19 19 19 19 17 17 10 12
7: 19 19 19 19 17 17 12 10

#redhat #rhsummit
8 (or 16) socket Ivy Bridge EX **prototype** server – NUMA topology

```bash
# numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 130956 MB
node 0 free: 125414 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 131071 MB
node 1 free: 126712 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 131072 MB
node 2 free: 126612 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 131072 MB
node 3 free: 125363 MB
node 4 cpus: 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
node 4 size: 131072 MB
node 4 free: 126479 MB
node 5 cpus: 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
node 5 size: 131072 MB
node 5 free: 125298 MB
node 6 cpus: 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
node 6 size: 131072 MB
node 6 free: 126913 MB
node 7 cpus: 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
node 7 size: 131072 MB
node 7 free: 124509 MB
node distances:
node 0 0 1 2 3 4 5 6 7
  0: 10 16 30 30 30 30 30 30
  1: 16 10 30 30 30 30 30 30
  2: 30 10 16 30 30 30 30 30
  3: 30 30 10 16 30 30 30 30
  4: 30 30 30 10 16 30 30 30
  5: 30 30 30 30 10 30 30 30
  6: 30 30 30 30 30 10 30 30
  7: 30 30 30 30 30 30 10 30
NUMA performance considerations

• NUMA performance penalties from two main sources
  • Higher latency of accessing remote memory
  • Interconnect contention

• Processor threads and cores share resources
  • Execution units (between HT threads)
  • Cache (between threads and cores)
Automatic NUMA balancing strategies

• CPU follows memory
  • Reschedule tasks on same nodes as memory
• Memory follows CPU
  • Copy memory pages to same nodes as tasks/threads
• Both strategies are used by automatic NUMA balancing
  • Various mechanisms involved
  • Lots of interesting corner cases...
Automatic NUMA Balancing
Internals
Automatic NUMA balancing internals

• NUMA hinting page faults
• NUMA page migration
• Task grouping
• Fault statistics
• Task placement
• Pseudo-interleaving
NUMA hinting page faults

• Periodically, each task's memory is unmapped
  • Period based on run time, and NUMA locality
  • Unmapped “a little bit” at a time (chunks of 256MB)
  • Page table set to “no access permission” marked as NUMA pte

• Page faults generated as task accesses memory
  • Used to track the location of memory a task uses
    • Task may also have unused memory “just sitting around”
  • NUMA faults also drive NUMA page migration
NUMA page migration

- NUMA page faults are relatively cheap
- Page migration is much more expensive
  - ... but so is having task memory on the “wrong node”
- Quadratic filter: only migrate if page is accessed twice
  - From same NUMA node, or
  - By the same task
  - CPU number & low bits of pid in page struct
- Page is migrated to where the task is running
Fault statistics

• Fault statistics are used to place tasks (cpu-follows-memory)
• Statistics kept per task
• “Where is the memory this task is accessing?”
  • NUMA page faults counter per NUMA node
  • After a NUMA fault, account the page location
    • If the page was migrated, account the new location
  • Kept as a floating average
Types of NUMA faults

• Locality
  • “Local fault” - memory on same node as task
  • “Remote fault” - memory on different node than task

• Private vs shared
  • “Private fault” - memory accessed by same task twice in a row
  • “Shared fault” - memory accessed by different task than last time
## Fault statistics example

<table>
<thead>
<tr>
<th>numa_faults</th>
<th>Task A</th>
<th>Task B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>0</td>
<td>1027</td>
</tr>
<tr>
<td>Node 1</td>
<td>83</td>
<td>29</td>
</tr>
<tr>
<td>Node 2</td>
<td>915</td>
<td>17</td>
</tr>
<tr>
<td>Node 3</td>
<td>4</td>
<td>31</td>
</tr>
</tbody>
</table>
Task placement

• Best place to run a task
  • Where most of its memory accesses happen
Task placement

• Best place to run a task
  • Where most of its memory accesses happen

• It is not that simple
  • Tasks may share memory
    • Some private accesses, some shared accesses
    • 60% private, 40% shared is possible – group tasks together for best performance
  • Tasks with memory on the node may have more threads than can run in one node's CPU cores
  • Load balancer may have spread threads across more physical CPUs
    • Take advantage of more CPU cache
Task placement constraints

• NUMA task placement may not create a load imbalance
  • The load balancer would move something else
  • Conflict can lead to tasks “bouncing around the system”
    • Bad locality
    • Lots of NUMA page migrations

• NUMA task placement may
  • Exchange tasks between nodes
  • Move a task to an idle CPU if no imbalance is created
Task placement algorithm

• For task a, check each NUMA node N
  • Check whether node N is better than task a's current node (C)
    • Task A has a larger fraction of memory accesses on node N, than on current node C
    • Score is the difference of fractions
  • If so, for each CPU on node N
    • Is the CPU idle and can we move task a to the CPU?
    • If not idle, is the current task (t) on CPU better off on node C?
    • Is the benefit of moving task a to node N larger than the downside of moving task t to node C?
  • For the CPU with the best score, move task a (and task t, to node C).
Task placement examples

<table>
<thead>
<tr>
<th>NODE</th>
<th>CPU</th>
<th>TASK</th>
<th>NODE 0</th>
<th>Fault statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a</td>
<td>30% (*)</td>
<td>Task a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>t</td>
<td>60% (*)</td>
<td>Task t</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(idle)</td>
<td>NODE 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
<td>70%</td>
<td>40%</td>
</tr>
</tbody>
</table>

• Moving task a to node 1: 40% improvement
• Moving task a to node 1 removes a load imbalance
• Moving task a to an idle CPU on node 1 is desirable
Task placement examples

<table>
<thead>
<tr>
<th>NODE</th>
<th>CPU</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(idle)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fault statistics</th>
<th>Task a</th>
<th>Task t</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE 0</td>
<td>30% (*)</td>
<td>60%</td>
</tr>
<tr>
<td>NODE 1</td>
<td>70%</td>
<td>40% (*)</td>
</tr>
</tbody>
</table>

• Moving task a to node 1: 40% improvement
• Moving task t to node 0: 20% improvement
• Exchanging tasks a & t is desirable
Task placement examples

<table>
<thead>
<tr>
<th>NODE</th>
<th>CPU</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(idle)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
</tr>
</tbody>
</table>

Fault statistics

<table>
<thead>
<tr>
<th>NODE</th>
<th>Task a</th>
<th>Task t</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE 0</td>
<td>30% (*)</td>
<td>40%</td>
</tr>
<tr>
<td>NODE 1</td>
<td>70%</td>
<td>60% (*)</td>
</tr>
</tbody>
</table>

• Moving task a to node 1: 40% improvement
• Moving task t to node 0: 20% worse
• Exchanging tasks a & t: overall a 20% improvement ==> do it
### Task placement examples

<table>
<thead>
<tr>
<th>NODE</th>
<th>CPU</th>
<th>TASK</th>
<th>Fault statistics</th>
<th>Task a</th>
<th>Task t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a</td>
<td>NODE 0</td>
<td>30% (*)</td>
<td>20%</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(idle)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>t</td>
<td>NODE 1</td>
<td>70%</td>
<td>80% (*)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Moving task a to node 1: 40% improvement
- Moving task t to node 0: 60% worse
- Exchanging tasks a & t: overall 20% worse ==> leave things alone
Task grouping

• Multiple tasks can access the same memory
  • Threads in a large multi-threaded process (JVM, virtual machine, ...)
  • Processes using shared memory segment (eg. Database)
• Use CPU num & pid in struct page to detect shared memory
  • At NUMA fault time, check CPU where page was last faulted
  • Group tasks together in numa_group, if PID matches
• Grouping related tasks improves NUMA task placement
  • Only group truly related tasks
  • Only group on write faults, ignore shared libraries like libc.so
Task grouping & task placement

• Group stats are the sum of the NUMA fault stats for tasks in group
• Task placement code similar to before
• If a task belongs to a numa_group, use the numa_group stats for comparison instead of the task stats
  • Pulls groups together, for more efficient access to shared memory
• When both compared tasks belong to the same numa_group
  • Use task stats, since group numbers are the same
  • Efficient placement of tasks within a group
Task grouping & placement example
Task grouping & placement example
Pseudo-interleaving

• Sometimes one workload spans multiple nodes
  • More threads running than one node has CPU cores
  • Spread out by the load balancer

• Goals
  • Maximize memory bandwidth available to workload
  • Keep private memory local to tasks using it
  • Minimize number of page migrations
Pseudo-interleaving problem

• Most memory on node 0, sub-optimal use of memory bandwidth
• How to fix? Spread out the memory more evenly...
Pseudo-interleaving algorithm

• Determine nodes where workload is actively running
  • CPU time used & NUMA faults
• Always allow private faults (same task) to migrate pages
• Allow shared faults to migrate pages only from a more heavily used node, to a less heavily used node
• Block NUMA page migration on shared faults from one node to another node that is equally or more heavily used
Pseudo-interleaving solution

- Allow NUMA migration on private faults
- Allow NUMA migration from more used, to lesser used node
Pseudo-interleaving converged state

- Nodes are equally used, maximizing memory bandwidth
- NUMA page migration only on private faults
- NUMA page migration on shared faults is avoided
Automatic NUMA Placement Performance

Show me the numbers!
Evaluation of Automatic NUMA balancing – Status update

Goal: Study the impact of Automatic NUMA Balancing on out-of-the-box performance compared to no NUMA tuning and manual NUMA pinning

• On bare-metal and KVM guests
• Using a variety of synthetic workloads*
  • 2 Java workloads
     • SPECjbb2005 used as a workload
     • Multi-JVM server workload
  • Database
     • A synthetic DSS workload (using tmpfs)
     • A synthetic OLTP workload in KVM (using virtio)

* Note: These sample workloads were used for relative performance comparisons. This is not an official benchmarking exercise!
Experiments with bare-metal

- Platforms used:
  - HP Proliant DL580 Gen 8 - 4-socket Ivy Bridge EX server
  - 8-socket Ivy Bridge EX prototype server.

- Misc. settings:
  - Hyper-threading off, THP enabled & cstate set to 1

- Configurations:
  - Baseline: No manual pinning of the workload, No Automatic NUMA balancing
  - Pinned: Manual (numactl) pinning of the workload
  - Automatic NUMA balancing: default, out-of-the-box setting.
Tools

• Status of Automatic NUMA balancing
  • Use sysctl to check/disable/enable “kernel.numa_balancing”
  • Default is set to enabled.

• /proc/vmstat
  • Indication of # of pages migrated & # of pages that failed to migrate

• /proc/zoneinfo
  • Indication of remote vs. local NUMA accesses

• numastat
  • Indication of which nodes are contributing to the running tasks.

• Miscellaneous upstream tools : e.g. numatop
SPECjbb2005 - bare-metal
(4-socket IVY-EX server vs. 8-socket IVY-EX prototype server)

1s wide = 15 warehouse threads, 2s wide = 30 warehouse threads; 4s wide = 60 warehouse threads, 8s wide = 120 warehouse threads

Pinned case was ~10-25% better than Baseline case
Automatic NUMA balancing case &
the Pinned case were pretty close (+/- 4%).

Pinned case was ~34-65% better than the Baseline case.
Delta between Automatic NUMA balancing case &
the Pinned case was as high as ~18%.
Remote vs. local memory access (RMA/LMA samples)*
(Workload: Multiple 1 socket-wide instances of SPECjbb2005)

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>33142</td>
<td>java</td>
<td>375281.8</td>
<td>131458.3</td>
<td>2.9</td>
<td>0.96</td>
<td>25.9</td>
</tr>
<tr>
<td>33140</td>
<td>java</td>
<td>373228.2</td>
<td>124433.0</td>
<td>3.0</td>
<td>1.00</td>
<td>25.8</td>
</tr>
<tr>
<td>33141</td>
<td>java</td>
<td>377242.2</td>
<td>121182.1</td>
<td>3.1</td>
<td>1.02</td>
<td>25.7</td>
</tr>
<tr>
<td>33143</td>
<td>java</td>
<td>344362.5</td>
<td>157420.1</td>
<td>2.2</td>
<td>0.98</td>
<td>25.7</td>
</tr>
</tbody>
</table>

Baseline

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>39678</td>
<td>java</td>
<td>200041.3</td>
<td>25753.2</td>
<td>8.1</td>
<td>3.09</td>
<td>13.0</td>
</tr>
<tr>
<td>39676</td>
<td>java</td>
<td>235889.3</td>
<td>344595.6</td>
<td>6.8</td>
<td>2.77</td>
<td>12.0</td>
</tr>
<tr>
<td>39680</td>
<td>java</td>
<td>224735.9</td>
<td>219336.6</td>
<td>10.5</td>
<td>3.61</td>
<td>12.0</td>
</tr>
<tr>
<td>39679</td>
<td>java</td>
<td>215382.5</td>
<td>251748.6</td>
<td>5.4</td>
<td>2.91</td>
<td>12.0</td>
</tr>
<tr>
<td>39678</td>
<td>java</td>
<td>235889.3</td>
<td>344595.6</td>
<td>6.8</td>
<td>2.77</td>
<td>12.0</td>
</tr>
<tr>
<td>39677</td>
<td>java</td>
<td>219294.3</td>
<td>252644.2</td>
<td>7.4</td>
<td>3.16</td>
<td>12.8</td>
</tr>
<tr>
<td>39675</td>
<td>java</td>
<td>197423.0</td>
<td>240237.6</td>
<td>6.1</td>
<td>2.44</td>
<td>12.8</td>
</tr>
<tr>
<td>39673</td>
<td>java</td>
<td>231181.8</td>
<td>45909.4</td>
<td>5.1</td>
<td>2.40</td>
<td>12.8</td>
</tr>
</tbody>
</table>

8-socket IVY-EX prototype server

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>25494</td>
<td>java</td>
<td>296.1</td>
<td>629725.2</td>
<td>5.0</td>
<td>0.88</td>
<td>14.5</td>
</tr>
<tr>
<td>25492</td>
<td>java</td>
<td>277.3</td>
<td>60745.1</td>
<td>0.0</td>
<td>0.67</td>
<td>14.3</td>
</tr>
<tr>
<td>25490</td>
<td>java</td>
<td>249.9</td>
<td>565102.3</td>
<td>0.0</td>
<td>0.68</td>
<td>14.3</td>
</tr>
<tr>
<td>25489</td>
<td>java</td>
<td>244.1</td>
<td>501810.3</td>
<td>0.0</td>
<td>0.68</td>
<td>14.3</td>
</tr>
<tr>
<td>25488</td>
<td>java</td>
<td>233.4</td>
<td>605667.1</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25487</td>
<td>java</td>
<td>208.7</td>
<td>617447.8</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25486</td>
<td>java</td>
<td>271.0</td>
<td>595257.3</td>
<td>0.0</td>
<td>0.88</td>
<td>14.3</td>
</tr>
<tr>
<td>25485</td>
<td>java</td>
<td>68.5</td>
<td>607569.8</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
</tbody>
</table>

Pinned

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>31769</td>
<td>java</td>
<td>104.5</td>
<td>548278.1</td>
<td>0.0</td>
<td>0.78</td>
<td>26.0</td>
</tr>
<tr>
<td>31768</td>
<td>java</td>
<td>118.4</td>
<td>565379.1</td>
<td>0.0</td>
<td>0.77</td>
<td>25.9</td>
</tr>
<tr>
<td>31771</td>
<td>java</td>
<td>64.1</td>
<td>543875.8</td>
<td>0.0</td>
<td>0.77</td>
<td>25.9</td>
</tr>
<tr>
<td>31770</td>
<td>java</td>
<td>138.1</td>
<td>545564.5</td>
<td>0.0</td>
<td>0.78</td>
<td>25.8</td>
</tr>
</tbody>
</table>

Baseline

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>25494</td>
<td>java</td>
<td>296.1</td>
<td>629725.2</td>
<td>5.0</td>
<td>0.88</td>
<td>14.5</td>
</tr>
<tr>
<td>25492</td>
<td>java</td>
<td>277.3</td>
<td>60745.1</td>
<td>0.0</td>
<td>0.67</td>
<td>14.3</td>
</tr>
<tr>
<td>25490</td>
<td>java</td>
<td>249.9</td>
<td>565102.3</td>
<td>0.0</td>
<td>0.68</td>
<td>14.3</td>
</tr>
<tr>
<td>25489</td>
<td>java</td>
<td>244.1</td>
<td>501810.3</td>
<td>0.0</td>
<td>0.68</td>
<td>14.3</td>
</tr>
<tr>
<td>25488</td>
<td>java</td>
<td>233.4</td>
<td>605667.1</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25487</td>
<td>java</td>
<td>208.7</td>
<td>617447.8</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25486</td>
<td>java</td>
<td>271.0</td>
<td>595257.3</td>
<td>0.0</td>
<td>0.88</td>
<td>14.3</td>
</tr>
<tr>
<td>25485</td>
<td>java</td>
<td>68.5</td>
<td>607569.8</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
</tbody>
</table>

Pinned

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>32464</td>
<td>java</td>
<td>535.5</td>
<td>556543.3</td>
<td>0.0</td>
<td>0.78</td>
<td>26.3</td>
</tr>
<tr>
<td>32453</td>
<td>java</td>
<td>572.5</td>
<td>548304.8</td>
<td>0.0</td>
<td>0.79</td>
<td>26.2</td>
</tr>
<tr>
<td>32452</td>
<td>java</td>
<td>33014.9</td>
<td>518331.7</td>
<td>0.0</td>
<td>0.80</td>
<td>26.2</td>
</tr>
<tr>
<td>32451</td>
<td>java</td>
<td>32145.0</td>
<td>507227.0</td>
<td>0.0</td>
<td>0.80</td>
<td>26.1</td>
</tr>
</tbody>
</table>

Automatic NUMA balancing

<table>
<thead>
<tr>
<th>PID</th>
<th>PROC</th>
<th>RMA (K)</th>
<th>LMA (K)</th>
<th>RMA/LMA</th>
<th>CPI</th>
<th>*CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>26586</td>
<td>java</td>
<td>120637.5</td>
<td>342139.0</td>
<td>8.4</td>
<td>1.13</td>
<td>13.4</td>
</tr>
<tr>
<td>26585</td>
<td>java</td>
<td>19826.7</td>
<td>514351.1</td>
<td>0.0</td>
<td>0.91</td>
<td>13.4</td>
</tr>
<tr>
<td>26584</td>
<td>java</td>
<td>159940.0</td>
<td>317038.3</td>
<td>0.5</td>
<td>1.13</td>
<td>13.4</td>
</tr>
<tr>
<td>26583</td>
<td>java</td>
<td>105379.4</td>
<td>377078.7</td>
<td>0.2</td>
<td>1.00</td>
<td>12.5</td>
</tr>
<tr>
<td>26582</td>
<td>java</td>
<td>75058.3</td>
<td>399095.2</td>
<td>0.2</td>
<td>1.01</td>
<td>12.5</td>
</tr>
<tr>
<td>26581</td>
<td>java</td>
<td>48782.9</td>
<td>439377.6</td>
<td>0.1</td>
<td>0.96</td>
<td>12.8</td>
</tr>
<tr>
<td>26580</td>
<td>java</td>
<td>48514.7</td>
<td>443236.5</td>
<td>0.1</td>
<td>0.96</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Automatic NUMA balancing

* Courtesy numatop v1.0.2

Higher RMA/LMA

#redhat #rhusummit
Multi-JVM server workload – bare-metal
(4-socket IVY-EX server vs. 8-socket IVY-EX prototype server)

Entities within each of the multiple Groups communicate with a Controller (using IPC) within the same host &
the frequency of communication increases as the # of Groups increase

Two key metrics: Max throughput (max-OPS) with no constraints & Critical throughput (critical-OPS) under fixed SLA constraints

Some workloads will still need manual pinning!
Database workload - bare-metal
(4-socket IVY-EX server)

Synthetic DSS workload (using tmpfs)
10GB Database size

~9-18% improvement in Average transactions per second with Automatic NUMA balancing
KVM guests

• Virtual machines are getting larger and larger
  • Use case 1: classic enterprise scale-up VMs
  • Use case 2: VMs in private cloud environments
• Low overhead & predictable performance for VMs (of any size) => careful planning & provisioning
  • Host's NUMA topology & the current resource usage
  • Manual pinning/tuning using libvirt/virsh
• Problem: It's harder to live migrate a pinned VM
  • Resources not available on destination host
  • Destination host - different NUMA topology!

```
<cpuset>
  <vcpupin vcpu='0' cpuset='0'/>
  <vcpupin vcpu='1' cpuset='1'/>
  ...
  <vcpupin vcpu='29' cpuset='29'/>
</cpuset>

<numatune>
  <memory mode='preferred' nodeset='0-1'/>
</numatune>
```
KVM guests (cont.)

• Automatic NUMA balancing avoids need for manual pinning
  • Exceptions - where memory pages can't be migrated
    • hugetlbfs (instead of THPs)
    • Device assignment (get_user_pages())
    • Hosting applications with real time needs (mlock_all()).
• VM >1 socket - enable Virtual NUMA nodes
  • Helps guest OS to scale/perform
  • Automatic NUMA balancing is enabled
  • Can pin the workload to virtual NUMA nodes

```xml
<cpu>
  <topology sockets='2' cores='15' threads='1'/>
  <numa>
    <cell cpus='0-14' memory='134217728'/>
    <cell cpus='15-29' memory='134217728'/>
  </numa>
</cpu>
```
KVM guests (cont.)

**Baseline VM**
- 8VCPU & 32GB VM
- 4-socket host
- 4 cores & 32 GB / socket
- VCPUs could run anywhere
- Backing memory could be remote
- No virtual NUMA nodes in the guest

**Pinned VM**
- Two 4VCPU/16GB virtual NUMA nodes
- Each VCPU pinned to each Host CPU core on specified sockets/nodes.
- Backing memory from specified Host nodes (strict/preferred/interleave)
- Virtual NUMA nodes => Better scaling in the guest OS

**Automatic NUMA bal. VM**
- Two 4VCPU/16GB virtual NUMA nodes
- Host OS’s Automatic NUMA bal. in control of finding the best backing resources for the VM
- Virtual NUMA nodes =>
  - Better scaling in the guest OS
  - Automatic NUMA bal. in Guest OS.
  - Can also pin workload in the Guest OS
KVM guests - HP Proliant DL 580 Gen8 - 4-socket Ivy Bridge EX server

• Guest sizes
  • 1s-wide guest → 15VCPUs/128GB
  • 2s-wide guest → 30VCPUs/256GB (2 virtual NUMA nodes)
  • 4s-wide guest → 60VCPUs/512GB (4 virtual NUMA nodes)

• Configurations tested
  • **Baseline VM** => a typical public/private cloud VM today
    • No pinning, no virtual NUMA nodes, no Automatic NUMA balancing in host or guest
  • **Pinned VM** => a typical enterprise scale-up VM today
    • VCPUs and memory pinned, virtual NUMA nodes (for > 1s wide VM)
    • Workload pinned in the guest OS (to virtual NUMA nodes)
  • **Automatic NUMA balanced VM** => “out of box” for any type of VM
    • Automatic NUMA balancing in host and guest
    • Virtual NUMA nodes enabled in the VM (for > 1s wide VM)
SPECjbb2005 in KVM
(4 socket IVY-EX server)

Pinned VM performed 5-16% better than Baseline VM.
Automatic NUMA bal. VM & the Pinned VM were pretty close (+/- 3%).
Multi-JVM server workload in KVM
(4-socket IVY-EX server)

Pinned VM was 11-18% better for max-OPS and 24-26% better for critical-OPS relative to the Baseline VM. For VMs up to 2 socket wide the Automatic NUMA bal. VM was closer to Pinned VM.

Delta between the Automatic NUMA bal. VM case & the Pinned VM case was much higher (~14% max-OPS and ~24% of critical-OPS)

Pinning the workload to the virtual NUMA nodes in the larger Automatic NUMA bal. Guest OS does bridge the gap.
KVM – server consolidation example 1
(Two VMs each running a different workload hosted on 4 Socket IVY-EX server)

**Synthetic DSS workload (using tmpfs)**

- 10GB Database size

**Baseline VM**

- Pinned VM

**Automatic NUMA bal. VM**

**# of users**

- 100
- 200
- 400

**Average Transactions per second**

**Operations per second**

**SPECjbb2005**

- 30VCPU/256GB

**# of warehouse threads**

- 30 warehouses

**Pinned VM was at least 10% better than Baseline VM.**

**Automatic NUMA bal. VM & the Pinned VM were pretty close (~ +/- 1-2%).**
KVM – server consolidation example 2
(Two VMs each running a different workload hosted on 4 Socket IVY-EX server)

For smaller VM size Automatic NUMA balancing was ~10-25% lower than Pinned VM
For larger VM size Automatic NUMA balancing was 5-15% lower than Baseline case!
NUMA Tools

What can I do?
NUMA Tools

• Numactl
• Numad
• taskset
• NUMA statistics in /proc
• Red Hat Enterprise Virtualization
numactl

• Control NUMA policy for processes or shared memory
  • numactl --arguments <program> <program arguments>
  • Bind to a node, interleave, ...
  • numactl --shmid to change properties of shared memory segment
• Show NUMA properties of the system
  • numactl --hardware
numad

• Optional user level daemon to do NUMA balancing for workloads or KVM guests
• More static than in-kernel automatic NUMA balancing
  • Better for some workloads
  • Worse for others
• Available in RHEL 6 & RHEL 7
  • You can use it today
Taskset

- Retrieve or set a process's CPU affinity
- Works on new commands, or PIDs of already running tasks
- Can bind tasks to the CPUs in a NUMA node
- Works for whole processes, or individual threads
NUMA statistics in /proc

- /proc/vmstat numa_* fields
  - NUMA locality (hit vs miss, local vs foreign)
  - Number of NUMA faults & page migrations
- /proc/<pid>/numa_maps
  - Location of process memory in NUMA system
- /proc/<pid>/sched
  - Numa scans, migrations & numa faults by node
Red Hat Enterprise Virtualization
(A preview of enhancements in the pipeline...)

- Graphical tool for creating KVM guests with NUMA awareness
- Visualize host NUMA topology and current resource usage.
- Define Virtual NUMA nodes for a guest.
- Bind to NUMA nodes on the host (optional)
- Will work with RHEL6 & RHEL7 based RHEV-H hypervisors

(final version of the GUI will differ)
Future Developments

What can't it do (yet)?
NUMA balancing future considerations

• Complex NUMA topologies & pseudo-interleaving
• Unmovable memory
• KSM
• Interrupt locality
• Inter Process Communication
Complex NUMA topologies & pseudo-interleaving

• Differing distances between NUMA nodes
  • Local node, nearby nodes, far away nodes
  • Eg. 20% & 100% performance penalty for nearby vs. far away

• Workloads that are spread across multiple nodes work better when those nodes are near each other

• Prototype implementation written
  • Different topologies need different placement algorithms
Backplane controller NUMA topology

• Backplane node controllers (XNC) in-between groups of nodes
• HP DL980
Backplane topology placement policy

• Compare nodes A and B
• If A and B are in different groups
  • Add the faults from all the nodes in each group
  • Migrate task to B if the group containing B has a higher score
  • Subject to load balancing & score constraints
• With A and B at N hops away from each other, each group consists of the nodes <N hops away from A or B
Glueless NUMA topology

- Traffic may bounce via intermediary node(s)
- Eg. Intel QPI links
- Fujitsu Primequest
Glueless NUMA topology

• Compare nodes A and B
• For each node A and B
  • Count faults on each node normally
  • Add faults from nodes 1 hop away, divided by 2
  • Add faults from nodes 2 hops away, divided by 4
  • Etc... (skip the furthest-away nodes for efficiency)
• Migrate tasks to node with the highest score
  • Subject to load balancing and score constraints
NUMA balancing & unmovable memory

• Unmovable memory
  • Mlock
  • Hugetlbfs
  • Pinning for KVM device assignment & RDMA

• Memory is not movable ...
  • But the tasks are
  • NUMA faults would help move the task near the memory
  • Unclear if worthwhile, needs experimentation
KSM

• Kernel Samepage Merging
  • De-duplicates identical content between KVM guests
  • Also usable by other programs

• KSM has simple NUMA option
  • “Only merge between tasks on the same NUMA node”
  • Task can be moved after memory is merged
  • May need NUMA faults on KSM pages, and re-locate memory if needed
  • Unclear if worthwhile, needs experimentation
Interrupt locality

• Some tasks do a lot of IO
  • Performance benefit to placing those tasks near the IO device
  • Manual binding demonstrates that benefit
• Currently automatic NUMA balancing only does memory & CPU
• Would need to be enhanced to consider IRQ/device locality
• Unclear how to implement this
  • When should IRQ affinity outweigh CPU/memory affinity?
Inter Process Communication

• Some tasks communicate a LOT with other tasks
  • Benefit from NUMA placement near tasks they communicate with
  • Do not necessarily share any memory
• Loopback TCP/UDP, named socket, pipe, ...
• Unclear how to implement this
  • How to weigh against CPU/memory locality?
Conclusions

• NUMA systems are increasingly common
• Automatic NUMA balancing improves performance for many workloads, on many kinds of systems
  • Can often get within a few % of optimal manual tuning
  • Manual tuning can still help with certain workloads
• Future improvements may expand the range of workloads and systems where automatic NUMA placement works nearly optimal