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
• 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
4 socket Ivy Bridge EX server – NUMA topology

# 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
8 socket Ivy Bridge EX **prototype** server – NUMA topology

```
# 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: 125383 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 1 2 3 4 5 6 7
0: 10 16 30 30 30 30 30 30 30 30 30 30 30 30 30
1: 16 10 30 30 30 30 30 30 30 30 30 30 30 30 30
2: 30 30 10 30 30 30 30 30 30 30 30 30 30 30 30
3: 30 30 10 30 30 30 30 30 30 30 30 30 30 30 30
4: 30 30 30 10 30 30 30 30 30 30 30 30 30 30 30
5: 30 30 30 30 10 30 30 30 30 30 30 30 30 30 30
6: 30 30 30 30 30 10 30 30 30 30 30 30 30 30 30
7: 30 30 30 30 30 30 10 30 30 30 30 30 30 30 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
  • Try running tasks where their memory is
• Memory follows CPU
  • Move memory to where it is accessed
• 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 tries to access 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, and per numa_group
• “Where is the memory this task (or group) 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 CPU
  • “Remote fault” - memory on different node than CPU

• 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
  • Swap 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, check all CPUs on node N
    • Is the current task (T) on CPU better off on node C?
    • Is the CPU idle, and can we move task A to the CPU?
    • 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(idle)</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 a task 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>
<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>60%</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>40%</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: 20% improvement
• Swapping tasks A & T is desirable
Task placement examples

<table>
<thead>
<tr>
<th>NODE</th>
<th>CPU</th>
<th>TASK</th>
<th>Fault statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>NODE 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(idle)</td>
<td>TASK A</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>T</td>
<td>TASK T</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
<td>NODE 1</td>
</tr>
</tbody>
</table>

Fault statistics:
- TASK A
  - NODE 0: 30% (*)
  - NODE 1: 70%
- TASK T
  - NODE 0: 40%
  - NODE 1: 60% (*)

- Moving task A to node 1: 40% improvement
- Moving task T to node 0: 20% worse
- Swapping 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>TASK A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(idle)</td>
<td>TASK T</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>T</td>
<td>NODE 0 30% (*)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>(idle)</td>
<td>NODE 1 70%</td>
</tr>
</tbody>
</table>

• Moving task A to node 1: 40% improvement
• Moving task T to node 0: 60% worse
• Swapping 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 (e.g. 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

Node 0

Node 1
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 on out-of-box performance with different workloads on bare-metal & KVM guests.

• Workloads being used*:
  • 2 Java workloads
    • SPECjbb2005 used as a workload
    • Multi-JVM server workload
  • Database
    • A synthetic DSS workload (using tmpfs)
    • Other DB workloads (with I/O) are in the pipeline...

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

• Platforms used:
  • 4-socket Ivy Bridge EX server
  • 8-socket Ivy Bridge EX prototype server.

• Misc. settings:
  • Hyper-threading off, THP enabled & cstates set to 1 (i.e. intel_idle.max_cstate=1 processor.max_cstate=1)

• Configurations:
  • Baseline:
    • No manual pinning of the workload
    • No Automatic NUMA balancing
  • Pinned:
    • Manual (numactl) pinning of the workload
  • Automatic NUMA balancing:
    • No manual pinning of the workload.
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

Automatic NUMA balancing case & the Pinned case were pretty close (+/- 4%).

Delta between Automatic NUMA balancing case & the Pinned case was as high as ~20%.
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>*CPI%</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>375322.6</td>
<td>124433.0</td>
<td>2.9</td>
<td>1.00</td>
<td>25.9</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>25.8</td>
</tr>
<tr>
<td>33143</td>
<td>java</td>
<td>343362.5</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>*CPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>39670</td>
<td>java</td>
<td>208041.3</td>
<td>25733.2</td>
<td>8.1</td>
<td>3.09</td>
<td>13.0</td>
</tr>
<tr>
<td>39671</td>
<td>java</td>
<td>235889.3</td>
<td>345885.6</td>
<td>6.8</td>
<td>2.77</td>
<td>13.0</td>
</tr>
<tr>
<td>39672</td>
<td>java</td>
<td>224735.9</td>
<td>21333.3</td>
<td>10.5</td>
<td>2.81</td>
<td>13.0</td>
</tr>
<tr>
<td>39673</td>
<td>java</td>
<td>215842.5</td>
<td>25174.8</td>
<td>7.4</td>
<td>2.91</td>
<td>13.0</td>
</tr>
<tr>
<td>39674</td>
<td>java</td>
<td>235901.2</td>
<td>32988.5</td>
<td>7.4</td>
<td>2.77</td>
<td>12.9</td>
</tr>
<tr>
<td>39675</td>
<td>java</td>
<td>219224.1</td>
<td>22228.7</td>
<td>7.5</td>
<td>2.80</td>
<td>12.8</td>
</tr>
<tr>
<td>39676</td>
<td>java</td>
<td>197423.0</td>
<td>24037.6</td>
<td>5.7</td>
<td>2.67</td>
<td>12.8</td>
</tr>
<tr>
<td>39677</td>
<td>java</td>
<td>231181.8</td>
<td>45808.4</td>
<td>5.5</td>
<td>2.52</td>
<td>12.8</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>*CPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>25494</td>
<td>java</td>
<td>296.1</td>
<td>62975.2</td>
<td>0.0</td>
<td>0.88</td>
<td>14.5</td>
</tr>
<tr>
<td>25495</td>
<td>java</td>
<td>277.3</td>
<td>60745.1</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25496</td>
<td>java</td>
<td>289.9</td>
<td>56310.2</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25497</td>
<td>java</td>
<td>244.1</td>
<td>50318.0</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25498</td>
<td>java</td>
<td>233.4</td>
<td>60567.1</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25499</td>
<td>java</td>
<td>200.7</td>
<td>61744.5</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25500</td>
<td>java</td>
<td>271.0</td>
<td>59525.7</td>
<td>0.0</td>
<td>0.88</td>
<td>14.4</td>
</tr>
<tr>
<td>25501</td>
<td>java</td>
<td>68.5</td>
<td>60796.8</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>*CPI%</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>543257.5</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>545664.5</td>
<td>0.0</td>
<td>0.78</td>
<td>25.9</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>*CPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>26588</td>
<td>java</td>
<td>120837.5</td>
<td>342139.0</td>
<td>8.4</td>
<td>1.13</td>
<td>13.4</td>
</tr>
<tr>
<td>26589</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>26590</td>
<td>java</td>
<td>159906.0</td>
<td>317038.3</td>
<td>0.0</td>
<td>1.13</td>
<td>13.4</td>
</tr>
<tr>
<td>26591</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>26592</td>
<td>java</td>
<td>103757.4</td>
<td>377078.7</td>
<td>0.0</td>
<td>1.00</td>
<td>10.9</td>
</tr>
<tr>
<td>26593</td>
<td>java</td>
<td>75605.6</td>
<td>399095.2</td>
<td>0.2</td>
<td>1.01</td>
<td>12.1</td>
</tr>
<tr>
<td>26594</td>
<td>java</td>
<td>48728.9</td>
<td>439377.9</td>
<td>0.1</td>
<td>0.96</td>
<td>12.1</td>
</tr>
<tr>
<td>26595</td>
<td>java</td>
<td>48514.7</td>
<td>443285.7</td>
<td>0.1</td>
<td>0.96</td>
<td>12.1</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>*CPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>39678</td>
<td>java</td>
<td>235889.3</td>
<td>345885.6</td>
<td>6.8</td>
<td>2.77</td>
<td>13.0</td>
</tr>
<tr>
<td>39679</td>
<td>java</td>
<td>224735.9</td>
<td>21333.3</td>
<td>10.5</td>
<td>2.81</td>
<td>13.0</td>
</tr>
<tr>
<td>39680</td>
<td>java</td>
<td>215842.5</td>
<td>25174.8</td>
<td>7.4</td>
<td>2.91</td>
<td>13.0</td>
</tr>
<tr>
<td>39681</td>
<td>java</td>
<td>235901.2</td>
<td>32988.5</td>
<td>7.4</td>
<td>2.77</td>
<td>12.9</td>
</tr>
<tr>
<td>39682</td>
<td>java</td>
<td>219224.1</td>
<td>22228.7</td>
<td>7.5</td>
<td>2.80</td>
<td>12.8</td>
</tr>
<tr>
<td>39683</td>
<td>java</td>
<td>197423.0</td>
<td>24037.6</td>
<td>5.7</td>
<td>2.67</td>
<td>12.8</td>
</tr>
<tr>
<td>39684</td>
<td>java</td>
<td>231181.8</td>
<td>45808.4</td>
<td>5.5</td>
<td>2.52</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Pinned

* Courtesy numatop v1.0.2

Higher RMA/LMA
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 trnphs)
10GB Database size

Static 2M huge pages used for SGA

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

• Size of the guests continue to increase
  • Use cases include classic enterprise scale-up guests & guests in private cloud environments.
  • [Manual] pinning/tuning of individual guests using libvirt/virsh (after taking into account host’s NUMA topology) are required to achieve low overhead & predictable performance for workloads running in mid/large sized guests. This is especially true on larger scale-up hosts (e.g. >= 4 socket servers)

    <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
    ...
    ...
    ...
    <vcpupin vcpu='29' cpuset='29'/>
    </cputune>

    <numatune>
    <memory mode='preferred' nodeset='0-1'/>
    </numatune>

• … but, its harder to live migrate such pinned guest across hosts – as similar set of backing resources may not be available on the target host (or) the target host may have a different NUMA topology!
KVM guests (contd.)

• Automatic NUMA balancing could help avoid the need for having to manually pin the guest resources.
  • Exceptions include cases where a guest's backing memory pages on the host can't be migrated:
    • Guests relying on hugetlbfs (instead of THPs)
    • Guests with direct device assignment (get_user_pages())
    • Guests with real time needs (mlock_all()).
  
• As the guest size spans more than 1 socket it is highly recommended to enable Virtual NUMA nodes in the guest => helps the guest OS instance to scale/perform.

```
<cpu>
  <topology sockets='2' cores='15' threads='1'/>
  <numa>
    <cell cpus='0-14' memory='134217728'/>
    <cell cpus='15-29' memory='134217728'/>
  </numa>
</cpu>
```

• Virtual NUMA nodes in the guest OS => Automatic NUMA balancing enabled in the guest OS instance.
  • Users can still choose to pin the workload to these Virtual NUMA nodes, if it helps their use case.
Experiments with KVM guests

• Platform used: 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: (different possible permutations, but we chose to focus on the following three)
  • Baseline guest (=> a typical public/private cloud guest today)
    • No pinning of VCPUs or memory,
    • No Virtual NUMA nodes enabled in the guest (even for >1s wide guest).
    • No Automatic NUMA balancing in Host or Guest OS.
    • Workload not pinned in the guest.
  • Pinned guest (=> a typical enterprise scale-up guest today)
    • VCPUs and memory pinned
    • Virtual NUMA nodes enabled in the guest (for >1s wide guest),
    • Workload is pinned in the guest (to the virtual NUMA nodes).
  • Automatic NUMA bal. guest: (=> “out of box” for any type of guest)
    • No manual pinning of VCPUs or memory,
    • Virtual NUMA nodes enabled in the guest (for >1s wide guest),
    • Automatic NUMA balancing in Host and Guest OS.
    • Workload not pinned in the guest. [a user could choose to pin a workload to the virtual NUMA nodes...without tripping over live migration issues.]
SPECjbb2005 in KVM guests
(4 socket IVY-EX server)

Automatic NUMA balancing case &
the Pinned guest case were pretty close (+/- 3%).
Multi-JVM server workload – KVM guests
(4-socket IVY-EX server)

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

Pinning the workload to the virtual NUMA nodes does help
KVM – server consolidation example
(Two 2-socket wide (30VCPU/256GB) guests each running a different workload hosted on 4 Socket IVY-EX server)

**Synthetic DSS workload (using tmpfs)**
- 10GB Database size

- # of users
  - 100
  - 200
  - 400

- Operations per second

![Graph showing performance comparison](image1)

**SPECjbb2005**
- # of warehouse threads
  - 30 warehouses

- Operations per second

![Graph showing performance comparison](image2)

*Automatic NUMA balancing case & the Pinned case were pretty close (~ 1-2%).*
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
• Unclear how to implement this
  • When is the second-best node no longer the second best?
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/memroy 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