KVM & Memory Management Updates

KVM Forum 2012

Rik van Riel
Red Hat, Inc.

KVM & Memory Management Updates

- EPT Accessed & Dirty Bits
- 1GB hugepages
- Balloon vs. Transparent Huge Pages
- Automatic NUMA Placement
- Automatic Ballooning

EPT Accessed & Dirty Bits

- Extended Page Tables (EPT)
    - Second set of page tables
    - Translates “guest physical” (virtual) to machine physical
        - Removes need for shadow page tables
- Originally, EPT only supported permissions and translations
    - Accessed & Dirty bits emulated in software
    - Take extra page faults to track A & D information
- EPT supports hardware Accessed & Dirty bit tracking in newer CPUs
    - Eliminates the extra page faults
- Already upstream

1GB hugepages

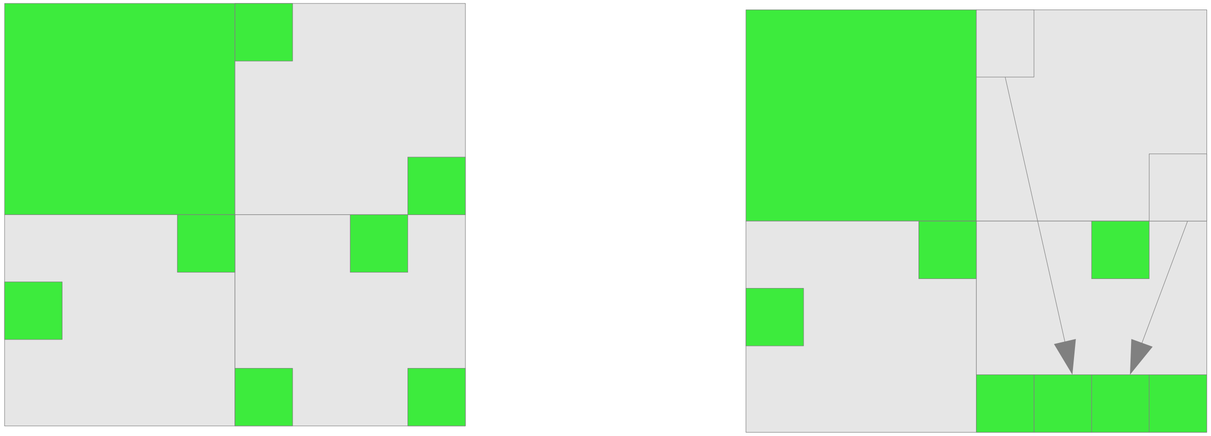
- Interface to hugetlbfs
    - Allows 1GB size huge pages
    - Desired page size can be specified at mmap time
    - Statically allocated
    - Not evictable, always pinned
    - In hugetlbfs only, not for transparent huge pages
- Use cases
    - HPC on bare metal
    - Use KVM for partitioning, with large guests
    - Want the last bit of performance
    - Does not need memory overcommit
- In -mm

Balloon vs. Transparent Huge Pages

- Transparent Huge Pages (THP)
  - Use 2MB pages for userspace when possible
  - Typical 5-15% performance improvement
  - Memory defragmented through compaction
    - Move data around, to free up 2MB area
- Balloon driver
  - Allocate memory in a guest
  - Return memory to the host
  - Guest does not use memory while in balloon
  - Effectively shrink guest memory size
- Proposed patch series by Rafael Aquini

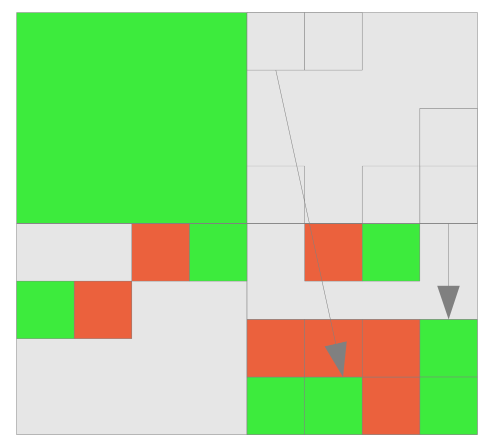
THP & Compaction

- Transparent Huge Pages (THP) needs 2MB blocks of memory
- Normal (4kB) allocations can fragment free memory
- Compaction can move page cache & anonymous memory data around
  - Frees up 2MB areas, for THP use



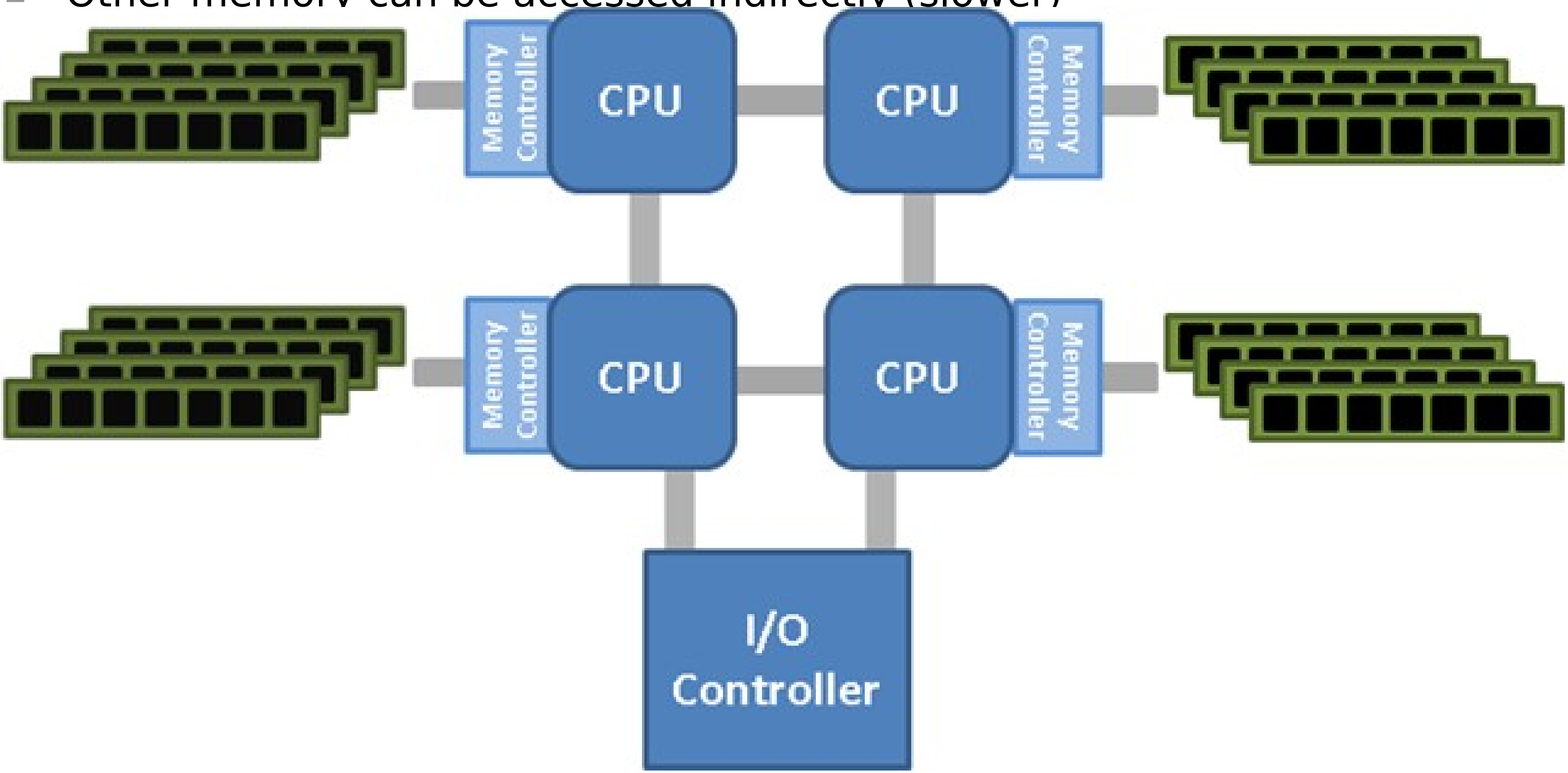
Compaction vs. Balloon pages

- Balloon pages are not anonymous or page cache memory
- The compaction code does not know how to move them
- Balloon pages can take a lot of memory
- When compaction fails, THP performance benefits not realized
- Teach compaction how to move balloon pages



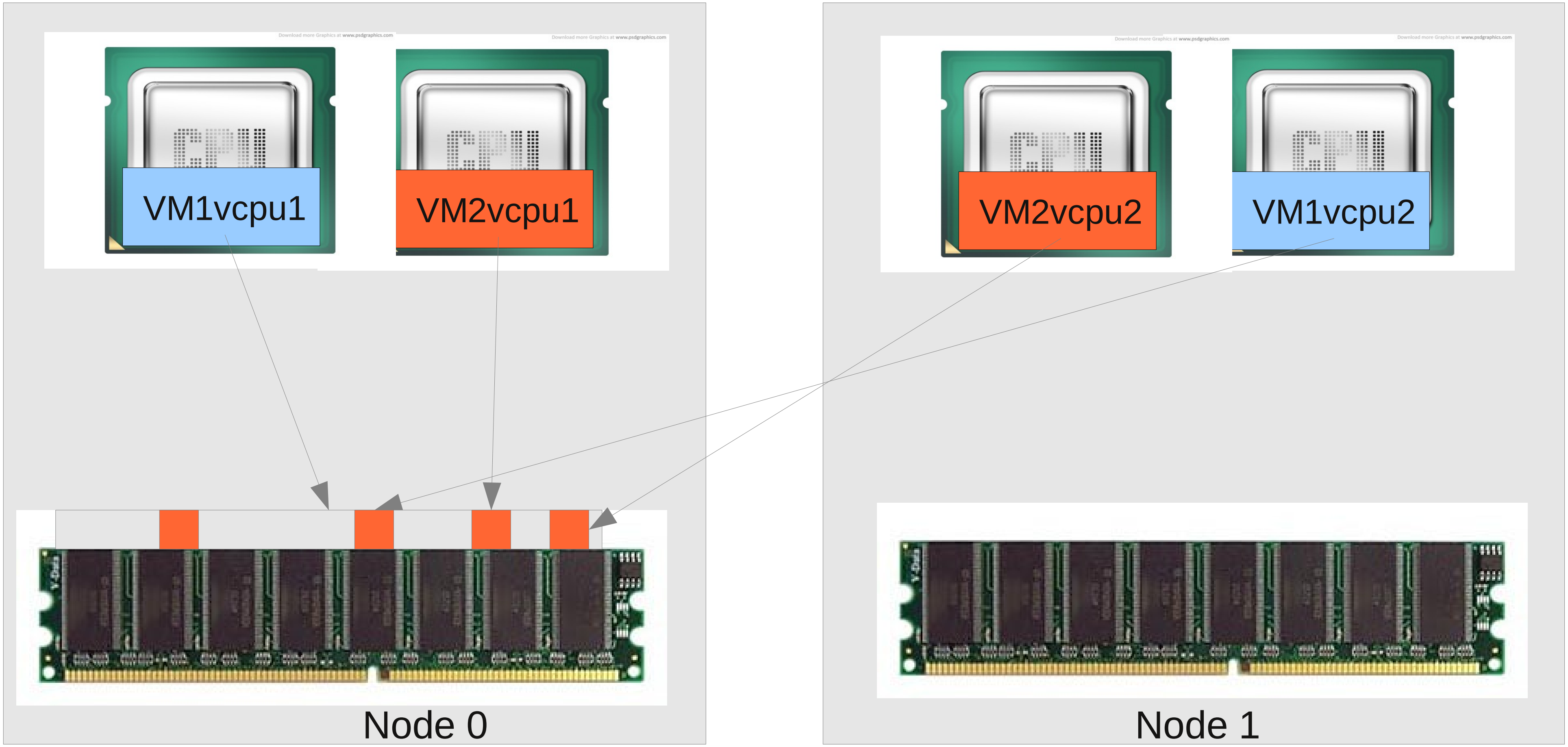
NUMA

- Non Uniform Memory Access
- Each CPU has its own memory (fast)
- Other memory can be accessed indirectly (slower)



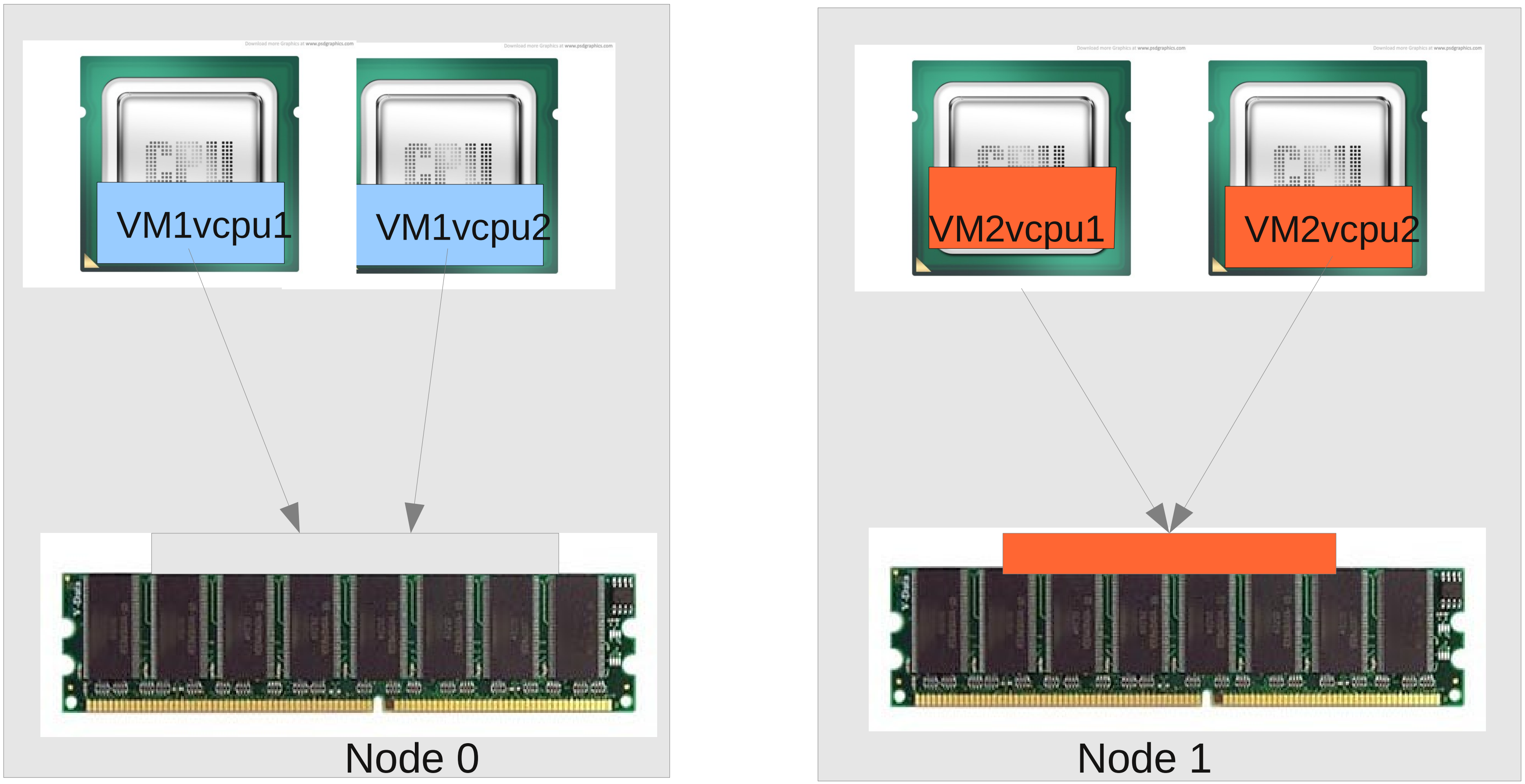
Unlucky NUMA Placement

- Without NUMA placement code, this can happen



Optimized NUMA Placement

- With numad, autonuma, sched/numa, numa/core, ...
- 3-15% performance improvement typical



Automatic NUMA Placement

- Obvious what better NUMA placement looks like, ...
- ... but how do we get there?
- Numad
  - Userspace NUMA placement daemon
  - For long lived tasks
  - Checks how much memory and CPU each task uses
  - “bin packing” to move tasks to NUMA nodes where they fit
  - Works right now
  - Not as good with dynamically changing workloads
  - More overhead, higher latency than kernel side solution
- Kernel side solution desired
- Several competing kernel side solutions proposed

Automatic NUMA Placement (Kernel)

- Three codebases:
  - Autonuma (Andrea Arcangeli)
  - Sched/numa & numa/core (Peter Zijlstra & Ingo Molnar)
  - Merged simple codebase (Mel Gorman)
- Some similarities
  - Strategies are mostly the same
  - NUMA Page Faults & Migration
  - Node scheduler affinity driven by fault statistics
- Some differences
  - Mostly implementation details

NUMA Faults & Migration

- Page fault driven NUMA migration
- “Memory follows CPU”
- Periodically, mark process memory as inaccessible
    - Clear present bit in page table entries, mark as NUMA
    - Rate limited to some number of MB/second
        - 3-15% NUMA gain typical, overhead limited to less
    - On “older” processes, short lived processes not affected
- When process tries to access memory
    - Page fault code recognizes NUMA fault
    - NUMA fault handling code is called
    - If page is on wrong NUMA node
        - Try to migrate to where the task is running now
    - If migrate fails, leave page on old NUMA node
        - No free memory on target node, page locked, ...
    - Increment per-task NUMA fault statistics for node where the faulted-on page is now

NUMA Faults & Migration

- Page fault driven NUMA migration
- "Memory follows CPU"
- Periodically, mark process memory as inaccessible
  - Clear present bit in page table entries, mark as NUMA
  - Rate limited to some number of MB/second
    - 3-15% NUMA gain typical, overhead limited to less
  - On "older" processes, short lived processes not affected
- When process tries to access memory
  - Page fault code recognizes NUMA fault
  - NUMA fault handling code is called
  - If page is on wrong NUMA node
    - Try to migrate to where the task is running now
  - If migrate fails, leave page on old NUMA node
    - No free memory on target node, page locked, ...
  - Increment per-task NUMA fault statistics for node where the faulted-on page is now

NUMA Fault Statistics

- NUMA page faults keep an array per task
  - Recent NUMA faults incurred on each node
  - Periodically the fault stats are divided by 2
    - Ages the stats, new faults count more than old ones
  - “Where is the memory I am currently using?”

	Faults
Node0	100
Node1	3200
Node2	5
Node3	0

Fault Driven Scheduler Affinity

- “CPU follows memory”
- Scheduler tries to run the task where its memory is
- Sched/numa & numa/core set desired task NUMA node
  - Hope the load balancer moves it later
- Autonuma searches other NUMA nodes
  - Node with more memory accesses than current node
    - Current task's NUMA locality must improve
  - Find task to trade places with
    - Overall cross-node accesses must go down
  - Tasks trade places immediately
    - Not using scheduler load balancer

Autonuma vs. Sched/numa vs. ...

- Autonuma
  - Performs better, but unknown exactly why
  - Actively groups threads within a task together
    - Fault stats not just per task, but also per mm
  - More complex than sched/numa or Mel's merged tree
    - Not known which complexity could be removed
- Sched/numa & numa/core
  - Simpler than autonuma
  - Does not perform as well
    - Large regressions on some tests, against mainline
  - Unclear what needs to be changed to make it work better
- Mel's merged tree
  - Clean, but extremely simplistic (MORON policy)
  - Basic infrastructure from autonuma and sched/numa
  - Meant as basis for a better placement policy
    - Clear way forward from this codebase

Dealing With NUMA Overflow

- Some workloads do not fit nicely on NUMA nodes
  - A process with more memory than on one NUMA node
  - A process with more threads than there are CPUs in a node
- Memory overflow
  - Not all of a process's memory fits on one node
  - Page migrations will fail
  - Fault statistics tell the scheduler where the memory is
  - Hopefully threads will migrate to their memory
- CPU overflow
  - Not all threads can run on one node
  - Memory gets referenced from multiple nodes
    - Only migrate on sequential faults from same node
  - Threads cannot get migrated to #1 preferred node
    - Migrate to nearby node instead
    - Migrate to node where process has many NUMA faults
- Unclear if this is enough...

Dealing With NUMA Overflow

- Some workloads do not fit nicely on NUMA nodes
  - A process with more memory than on one NUMA node
  - A process with more threads than there are CPUs in a node
- Memory overflow
  - Not all of a process's memory fits on one node
  - Page migrations will fail
  - Fault statistics tell the scheduler where the memory is
  - Hopefully threads will migrate to their memory
- CPU overflow
  - Not all threads can run on one node
  - Memory gets referenced from multiple nodes
    - Only migrate on sequential faults from same node
  - Threads cannot get migrated to #1 preferred node
    - Migrate to nearby node instead
    - Migrate to node where process has many NUMA faults
- Unclear if this is enough...

Memory Overcommit

- Cloud computing is a “race to the bottom”
- Cheaper and cheaper virtual machines expected
- Need to squeeze more virtual machines on each computer
    - CPU & bandwidth are easy
    - More CPU time and bandwidth become available each second
- Amount of memory stays constant (“non-renewable resource”)
- Swapping to disk is too slow
    - SSB is an option, but could be too expensive
    - Need something fast & free
    - Frontswap + zram an option
        - Compacting unused data slower than tossing it out
    - Ballooning guests is an option
        - Balloon driver returns memory from guest to host

Memory Overcommit

- Cloud computing is a “race to the bottom”
- Cheaper and cheaper virtual machines expected
- Need to squeeze more virtual machines on each computer
  - CPU & bandwidth are easy
  - More CPU time and bandwidth become available each second
- Amount of memory stays constant (“non-renewable resource”)
- Swapping to disk is too slow
  - SSB is an option, but could be too expensive
  - Need something fast & free
  - Frontswap + zram an option
    - Compacting unused data slower than tossing it out
  - Ballooning guests is an option
    - Balloon driver returns memory from guest to host

Automatic Ballooning

- Goal: squeeze more guests into a host, with minimal performance hit
  - Avoid swapping to disk
- Balloon driver is used to return memory to the host
  - For one guest to grow, others must give memory back
  - Could be done automatically
- Balloon guests down when host runs low on free memory
  - Put pressure on every guest
  - Guests return a little bit of memory to the host
  - Memory can be used by host itself, or by other guests
- Memory pressure inside guests deflates balloons
  - Guests get memory back from host in small increments
- Ideally, the pressures will balance out
- Some of the pieces are in place, some proposed, some not developed yet – longer term project

Vmevents & Guest Shrinking

- Proposed patch series by Anton Vorontsov, Pekka Engberg, ...
- Vmevents file descriptor
- Program opens fd using special syscall
- Can be blocking read() or poll/select()
- Kernel tells userspace when memory needs to be freed
  - Minimal pressure – free garbage collected memory?
  - Medium pressure – free some caches, shrink some guests?
  - OOM pressure – something unimportant should exit now
- Qemu-kvm, libvirtd or something else in virt stack could register for such vm events
  - When required, tell one or more guests to shrink
  - Guests will balloon some memory, freeing it to the host

Ballooning & Guest Growing

- Automatically shrinking guests is great ...
- ... but they may need their memory back at some point
- Linux pageout code has various pressure balancing concepts
  - #scanned / #rotated to see how heavily used pages in each LRU set are (“use ratio”)
  - Each cgroup has its own LRU sets, with its own use ratios
    - Pressure between cgroups independent of use ratios...
  - “seek cost” for objects in slab caches
  - Slab cache pressure independent of use ratio of LRU pages
  - ... this could use some unification
- The balloon can be called from the pageout code inside a guest
  - When we reclaim lots of pages ...
  - ... also request some pages from the balloon
  - Avoid doing this for streaming file I/O
- Adding pressure in the host, will result in being shrunk again later

Ballooning & QOS

- Ballooning can help avoid host swapping, which helps latencies
- But ...
  - Shrinking a guest too small could also impact latencies ...
  - ... or even cause a guest to run out of memory
- Never shrink a guest below a certain size
  - How big? No way to tell automatically...
- Reasonable minimum size needs to be specified in the configuration
- Minimums enforced by whatever listens for VM notifications
  - Qemu-kvm, libvirtd, vdsm, ... ?
- When a host gets overloaded, other actions need to be taken
  - Live migrate a guest away
  - Kill an unimportant guest?
- Automatic ballooning is part of a larger solution

Conclusions

- KVM performance continues to improve
  - No large gains left, but many small ones
- Uncontroversial changes (nearly) upstream
  - EPT A/D bits & 1GB huge pages
- Larger changes take time to get upstream
  - Complex problem? Unresolved questions...
- Two fiercely competing NUMA placement projects
  - Mel Gorman heroically smashed the two together
  - New tree looks like a good basis for moving forward
  - Typical 3-15% performance gains expected
- Squeeze more guests onto each system
  - Shrink and grow guest memory as needed
  - Extensive use of ballooning makes THP allocations harder
  - Patches proposed to make balloon pages movable

Questions?