Xen scheduler status

George Dunlap
Citrix Systems R&D Ltd, UK
george.dunlap@eu.citrix.com
Goals for talk

- Understand the problem: Why a new scheduler?
- Understand “reset events” in credit1 and credit2 algorithms
- Know design target and plans for future, so that you can participate
Outline

- Issues with current scheduler
- Design target for new scheduler
- Theory: Reset condition
  - Credit1 algorithm
  - Credit2 algorithm
- Future work
  - Load balancing
  - Hyperthreading
  - Power management
  - NUMA
What’s wrong with the old one?

- Client hypervisors and audio/video
  - Audio VM: 5% CPU
  - 2x Kernel-build VMs: 97% cpu each
  - 30-40 audio skips over 5 minutes
- Not fair to latency-sensitive workloads
  - Network scp: “Fair share” 50%, usage 20-30%
- Load balancing 64 threads (4 x 8 x 2)
  - Unpredictable
  - Not scalable
- Power management, Hyperthreads
What to aim for

- Xen use cases
  - Server consolidation
    - Key challenge: large number of vcpus
  - Virtual Desktop (VDI)
    - Key challenge: large number of VMs
  - Client virtualization (XenClient)
    - Key challenge: audio and video
Design goals, con’t

- **Evaluation criteria**
  - Fairness: Getting what you were promised
  - Throughput: Using all resources effectively
  - Graceful performance degradation

- **Issues to address**
  - Hyperthreading
    - Performance depends on what the other thread is doing
  - Power management
  - NUMA

- **Interface**
  - Reservation: Minimum CPU time
  - Weight: How to divide CPU when overcommitted
  - Cap: Maximum CPU time
Some theory

• Equivalence class of algorithms
  • A value per vcpu (credits, time, debits)
  • Value modified based on
    • Time running
    • Wall-clock time
    • Time blocked / on runqueue

• Key problem: Not all vcpus use all their time
  • Tendency towards divergence
Dealing with divergence

- Alternatives explored
  - Guessing how much credit would be used
  - Zero-sum: put in only as much as will be used
  - Simple cap

- Best solution: Reset event
  - Discards unused credits
  - Tends to “converge” vcpus who have gotten too far “behind”
  - Found in both Credit scheduler and BVT scheduler
Credit1 issues

- Many weaknesses
  - Long time-slice
  - Sorting by priority rather than credit
  - Probabilistic debiting
- Key issue: reset condition
Credit1: Core algorithm

- Two categories: active and inactive
- Active VMs
  - Credit divided every 30ms according to weight
  - Conceptually burn credits at a fixed rate
  - Two priorities: UNDER and OVER
  - Scheduling within a priority is round-robin
- Non-active VMs
  - Do not earn or burn credits
  - BOOST priority
- Transition
  - Active to Inactive: earn 30ms of credit
  - Inactive to active: interrupted by a tick (every 10ms)
Non-burner in Credit1

- Inactive is an unstable place to be
  - Guaranteed to be hit by a tick eventually
  - E.g. audio, 5% of cpu
  - 5% chance of getting hit by tick
  - Expect 1 in 20 ticks to hit the VM
  - Ticks every 10ms -> 200ms in “boost”

- Now in OVER
- Not burning all credits
  - Will go back to inactive after accumulating 30ms
- All vcpus using less than their “fair share” will flip back and forth between active and inactive
Credit2

- Reset condition is core to the algorithm
- Basic description
  - Credits for all VMs start at a fixed value
  - Credits consumed at different rates, based on weight
  - Insert into runqueue based on credits
- Reset condition:
  - When the credits of the vcpu at the front of the runqueue <= 0
  - Set everyone’s credits back to start value
  - No runqueue sorting required
- Refinement: Clip-and-add
  - Allows low-usage VMs to start at head of runqueue
  - Scheduler may allow a vcpu to go negative
Other Changes: Runqueue per L2

- Cache effects main reason to avoid migration
- Threads, cores share L2
- Instant “load balancing” across cores / threads shared by an L2
  - Including most single-socket boxes
Early results

- Audio is better
  - Same setup as before
  - 0 skips
- Network is more fair
  - Full 50% of CPU
Future work

- Load balancing
  - Only need between L2 runqueues
  - Heirarchical division
    - Linux “scheduling domain” concept
  - Explicit load balancing based on historical load
- Hyperthreading
  - Adjust “burn rate” for shared threads
- Power management
  - Weigh time waiting vs extra power
- NUMA
  - Weigh time waiting for cpu vs remote cache misses
Outline

- Issues with current scheduler
- Design target for new scheduler
- Theory: Reset condition
  - Credit1 algorithm
  - Credit2 algorithm
- Future work
  - Load balancing
  - Hyperthreading
  - Power management
  - NUMA
Goals for talk

- Understand the problem: Why a new scheduler?
- Understand “reset events” in credit1 and credit2 algorithms
- Know design target and plans for future, so that you can participate
Questions
Xen scheduler status

George Dunlap
Citrix Systems R&D Ltd, UK
george.dunlap@eu.citrix.com
Goals for talk

- Understand the problem: Why a new scheduler?
- Understand “reset events” in credit1 and credit2 algorithms
- Know design target and plans for future, so that you can participate
Outline

- Issues with current scheduler
- Design target for new scheduler
- Theory: Reset condition
  - Credit1 algorithm
  - Credit2 algorithm
- Future work
  - Load balancing
  - Hyperthreading
  - Power management
  - NUMA
What’s wrong with the old one?

- Client hypervisors and audio/video
  - Audio VM: 5% CPU
  - 2x Kernel-build VMs: 97% cpu each
  - 30-40 audio skips over 5 minutes
- Not fair to latency-sensitive workloads
  - Network scp: “Fair share” 50%, usage 20-30%
- Load balancing 64 threads (4 x 8 x 2)
  - Unpredictable
  - Not scalable
- Power management, Hyperthreads
What to aim for

- Xen use cases
  - Server consolidation
    - Key challenge: large number of vcpus
  - Virtual Desktop (VDI)
    - Key challenge: large number of VMs
  - Client virtualization (XenClient)
    - Key challenge: audio and video
Design goals, con’t

- Evaluation criteria
  - Fairness: Getting what you were promised
  - Throughput: Using all resources effectively
  - Graceful performance degradation

- Issues to address
  - Hyperthreading
    - Performance depends on what the other thread is doing
  - Power management
  - NUMA

- Interface
  - Reservation: Minimum CPU time
  - Weight: How to divide CPU when overcommitted
  - Cap: Maximum CPU time
Some theory

• Equivalence class of algorithms
  • A value per vcpu (credits, time, debits)
  • Value modified based on
    • Time running
    • Wall-clock time
    • Time blocked / on runqueue

• Key problem: Not all vcpus use all their time
  • Tendency towards divergence
Dealing with divergence

- Alternatives explored
  - Guessing how much credit would be used
  - Zero-sum: put in only as much as will be used
  - Simple cap

- Best solution: Reset event
  - Discards unused credits
  - Tends to “converge” vcpus who have gotten too far “behind”
  - Found in both Credit scheduler and BVT scheduler
Credit1 issues

- Many weaknesses
  - Long time-slice
  - Sorting by priority rather than credit
  - Probabilistic debiting
- Key issue: reset condition
Credit1: Core algorithm

- Two categories: active and inactive

- Active VMs
  - Credit divided every 30ms according to weight
  - Conceptually burn credits at a fixed rate
  - Two priorities: UNDER and OVER
  - Scheduling within a priority is round-robin

- Non-active VMs
  - Do not earn or burn credits
  - BOOST priority

- Transition
  - Active to Inactive: earn 30ms of credit
  - Inactive to active: interrupted by a tick (every 10ms)
Non-burner in Credit1

- Inactive is an unstable place to be
  - Guaranteed to be hit by a tick eventually
  - E.g. audio, 5% of cpu
  - 5% chance of getting hit by tick
  - Expect 1 in 20 ticks to hit the VM
  - Ticks every 10ms -> 200ms in “boost”

- Now in OVER

- Not burning all credits
  - Will go back to inactive after accumulating 30ms

- All vcpus using less than their “fair share” will flip back and forth between active and inactive
Credit2

- Reset condition is core to the algorithm
- Basic description
  - Credits for all VMs start at a fixed value
  - Credits consumed at different rates, based on weight
  - Insert into runqueue based on credits
- Reset condition:
  - When the credits of the vcpu at the front of the runqueue <= 0
  - Set everyone’s credits back to start value
  - No runqueue sorting required
- Refinement: Clip-and-add
  - Allows low-usage VMs to start at head of runqueue
  - Scheduler may allow a vcpu to go negative
Other Changes: Runqueue per L2

- Cache effects main reason to avoid migration
- Threads, cores share L2
- Instant “load balancing” across cores / threads shared by an L2
- Including most single-socket boxes
Early results

- Audio is better
  - Same setup as before
  - 0 skips

- Network is more fair
  - Full 50% of CPU
Future work

- Load balancing
  - Only need between L2 runqueues
  - Hierarchical division
    - Linux “scheduling domain” concept
  - Explicit load balancing based on historical load
- Hyperthreading
  - Adjust “burn rate” for shared threads
- Power management
  - Weigh time waiting vs extra power
- NUMA
  - Weigh time waiting for cpu vs remote cache misses
Outline

- Issues with current scheduler
- Design target for new scheduler
- Theory: Reset condition
  - Credit1 algorithm
  - Credit2 algorithm
- Future work
  - Load balancing
  - Hyperthreading
  - Power management
  - NUMA
Goals for talk

- Understand the problem: Why a new scheduler?
- Understand “reset events” in credit1 and credit2 algorithms
- Know design target and plans for future, so that you can participate
Questions