Consumption Policies and Unifying Heterogeneous Resource Constraints

Erik J. Erlandson
Red Hat, Inc.
eje@redhat.com
Agenda

● Goals
  - Introduce the new Consumption Policy feature
    - Available as of HTCondor 8.1.2
  - Describe how it can aid in thinking about:
    - accounting group quotas
    - match costs
    - slot weights

● Topics
  - Partitionable Slots
  - Scheduler splitting (CLAIM_PARTITIONABLE_LEFTOVERS)
  - Consumption Policies
  - Examples
  - Unit analysis for slot weights and match costs
In the Beginning: Partitionable Slots

• “p-slots” for short
• Present aggregate compute resources
• Designed to service multiple jobs
• Negotiator matches one job per p-slot per cycle
• Consequences
  - p-slots required multiple cycles to load
  - SlotWeight expressions make p-slots expensive
    - Accounting group starvation
Accounting Group Starvation

- Default: SlotWeight = Cpus
- SlotWeight on a 32-core machine = 32
  - Therefore cost to match = SlotWeight = 32
- An accounting group with quota < 32 can never match that resource
- This problem becomes more exaggerated as cores increase
- gittrac #3013
CLAIM_PARTITIONABLE_LEFTOVERS

- AKA “scheduler splitting”
- Side-step negotiator cycle bottleneck
- Enable scheduler to match multiple jobs against a p-slot matched in the negotiator

Limitations
- P-slot matches still expensive to the negotiator
  - Accounting group starvation still possible
- Doesn't play well with globally-accounted resources
  -Concurrency limits disrespected
- Matched resources not accessible to jobs from other schedulers
  - p-slot unavailable to negotiator until startd updates -> collector

Advantages:
- Improved scalability, especially with multiple schedulers
A Unit Analysis Question

- Suppose I have a pool where execute nodes advertise a mixture of slot weights:
  - SLOT_WEIGHT = Cpus
  - SLOT_WEIGHT = Memory
  - SLOT_WEIGHT = Disk

- When the negotiator computes the available resources by summing slot weights for all slots, what unit does that sum have?

- What unit do group quotas have?

- What does it mean to compare the cost of matching against one slot versus another?
Consumption Policies

- Resources consumed by a match between a job and a p-slot become a *configurable policy*
  - Expressions evaluated in context of p-slot resource classad
  - Special 'target' scope refers to candidate job classad
- Consumption Policy expressions reside on the p-slot classad
  - Available to startd claiming logic and negotiator matching logic
- Enable the negotiator to match multiple jobs against each p-slot in a single negotiation cycle
Matchmaking Flows

collector

p-slot

match
Job to
p-slot

deduct
resources

negotiator

updated p-slot

Claim
Leftovers

new claim

Consumption
Policy

match
Job to
p-slot

job

claim

claim

claim

deduct
resources

schedd

startd
A Simple Consumption Policy

# Assumes a partitionable slot configuration

# Enable use of consumption policies
CONSUMPTION_POLICY = True

# Define a simple consumption policy:
# "target" refers to the scope of the
# candidate job classad
CONSUMPTION_CPUS = target.RequestCpus
CONSUMPTION_MEMORY = target.RequestMemory
CONSUMPTION_DISK = target.RequestDisk

# Traditional CPU-centric match cost
SLOT_WEIGHT = Cpus
Match Cost With Consumption Policies

Recall: the legacy match cost = SlotWeight

Match cost for a p-slot with a consumption policy is defined as reduction in slot weight after deducting resources used by a match:

1. Evaluate SlotWeight (W)
   1. W <-- SlotWeight = Cpus = 8

2. Evaluate ConsumptionXXX expressions for each resource
   1. UsedCpus <-- ConsumptionCpus = target.RequestCpus = 1

3. Subtract consumed resources from p-slot resources
   1. Cpus <-- (Cpus – UsedCpus) = (8 – 1) = 7

4. Re-evaluate SlotWeight (W')
   1. W' <-- SlotWeight = Cpus = 7

5. Match cost = W – W'
   1. Cost <-- (W-W') = (8 – 7) = 1
Reusing P-Slots in the Negotiator

- Evaluate candidate match cost w.r.t. consumption policy expressions on the p-slot
- If resource consumption is not feasible, match fails: remove p-slot from the list
  - Insufficient resources
  - Failed to evaluate to integer values \(\geq 0\)
  - All consumption policies evaluated to zero
- If candidate match succeeds, subtract its resources and keep p-slot on the list
  - P-slot stays at front of list (depth-first loading)
- When slot weight drops to zero, remove from list
Pros and Cons

• Advantages
  – Negotiator can load p-slots in a single cycle
  – Concurrency limits respected
  – Jobs from multiple schedulers can match against a p-slot
  – Matches charged only for portion of resources used
    • Avoids accounting group starvation due to expensive p-slots

• Limitations
  – Negotiator bears cost of p-slot loading
    • Cannot scale out, as with scheduler splitting
Compatibility

• P-slots advertising a Consumption Policy can coexist with other slot flavors
  - P-slots having no consumption policy
  - Static slots
  - startds configured for CLAIM_PARTITIONABLE_LEFTOVERS
    • A startd cannot simultaneously enable consumption policies and leftovers
• Consumption Policies operate with extensible resources
  - A Consumption Policy expression must be declared for every resource, including extensible resources
    • All resources (including extensible) have default consumption policies
  - Not integrated with named (non-fungible) resources
Memory Centric Policy

CONSUMPTION_POLICY = True

CONSUMPTION_CPUS = target.RequestCpus
CONSUMPTION_MEMORY = quantize(target.RequestMemory, {128})
CONSUMPTION_DISK = quantize(target.RequestDisk, {1024})

# use of quantize() similar to MODIFY_REQUEST_EXPR_*

# synced with consumption expression
SLOT_WEIGHT = floor(Memory / 128)

# If total memory available is 1GB, then this
# slot + policy can support up to 8 matches, and
# total weight (prior to matching) is 8
Static Slot Policy

CONSUMPTION_POLICY = True

# consume all resources - emulate static slot
CONSUMPTION_CPUS = TotalSlotCpus
CONSUMPTION_MEMORY = TotalSlotMemory
CONSUMPTION_DISK = floor(0.9 * TotalSlotDisk)
# TotalSlotDisk != Disk even on an unused p-slot

# Slot supports exactly one match
SLOT_WEIGHT = 1
Multi-Centric Policy

CONSUMPTION_POLICY = True

# Either Cpus or Memory might be limiting
CONSUMPTION_CPUS = target.RequestCpus
CONSUMPTION_MEMORY = quantize(target.RequestMemory, {256})
CONSUMPTION_DISK = quantize(target.RequestDisk, {128})

# Define slot weight as minimum of remaining-match
# estimate based on either cpus or memory:
SLOT_WEIGHT = ifThenElse(Cpus < floor(Memory/256), Cpus, floor(Memory/256))

# Behaves a bit like Dominant Resource Fairness, due
# to submitter being effectively charged for the resource
# that most reduced the available matches against the p-slot
# ("Dominant Resource Fairshare")
Observations

- Match cost is defined as: reduction of slot weight after deducting resources used for a match

- The slot weight expression governs the orientation of the policy
  - SLOT_WEIGHT = Cpus
  - SLOT_WEIGHT = floor(Memory / 128)
  - SLOT_WEIGHT = floor(Disk / 1024)

- It also embodies a definition of how many matches the p-slot supports
  - If total memory available is 1 GB, then slot can support up to 8 matches
  - equivalent to number of jobs serviceable
Unifying Heterogeneous Policies

• A p-slot’s total slot weight is equivalent to the maximum number of matches it can support
  – i.e. Slot weights are in units of “matches”
  – This is true regardless of policy orientation: cpu-centric, memory-centric, etc
• Match cost = “reduction of slot weight” and is therefore in the same units: matches
• Assuming slot weights are enabled for matchmaking, then total resource assessment, and therefore accounting group quotas, are also in these same units
  – Particularly when configuring dynamic quotas
• Therefore: Slot weights, match cost and group quotas can be modeled in the same unit: matches (aka jobs, aka claims)
  – Furthermore, this unit analysis holds for pools combining p-slots having heterogeneous policy orientations
Future Development

• Non-integer resources
  – Model concepts such as sub-core jobs
• Integration with named (non-fungible) resources
  – GPUs
• Support breadth first p-slot loading
  – Currently, slots are loaded depth first
References

- http://research.cs.wisc.edu/htcondor/manual/v8.1/3_3Configuration.html#20322
- http://erikerlandson.github.io/blog/categories/slot-weights/