Arvados - Bug #12573

Priority is ignored in Crunch2

11/07/2017 08:07 PM - Tom Morris

Status:	Resolved	**Start date:**	11/30/2017
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2017-12-06 Sprint		

Description

As a workflow submitter, I would like to be able to set the priority on a workflow and have it influence the scheduling of jobs.

Priority runs from 1 (low) to 1000 (high). Priority 0 means canceled.

Components to be updated include:

- API Server
- Crunch dispatch slurm

A dispatcher configuration parameter is used to specify the range of niceness that is used to schedule things.

Subtasks:

Task # 12636: Fix	**Resolved**
Task # 12634: Review 12573-crunch2-slurm-priority	**Resolved**

Related issues:

Related to Arvados - Story #12552: When priority is equal, the children of an...	**Resolved**	**11/03/2017**

Associated revisions

Revision bcf069c6 - 12/04/2017 09:32 PM - Peter Amstutz

Merge branch '12573-crunch2-slurm-priority' closes #12573

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <pamstutz@veritasgenetics.com>

History

#1 - 11/15/2017 07:42 PM - Tom Clegg

For the crunch-dispatch-slurm case: Suggest submitting each job with nice=10000, monitoring the actual priority assigned by slurm, and using scontrol to increase the nice value such that the priority is what we want.

```
$ squeue -o '%i %y %Q'
JOBID NICE PRIORITY
104261 5000 4294874295
$ scontrol update jobid=104261 nice=2000
$ squeue -o '%i %y %Q'
JOBID NICE PRIORITY
104261 2000 4294877295
```

#2 - 11/15/2017 07:54 PM - Tom Morris

- Description updated

- Story points set to 1.0

#3 - 11/22/2017 07:38 PM - Tom Morris

- Target version changed from To Be Groomed to 2017-12-06 Sprint

#4 - 11/22/2017 07:54 PM - Peter Amstutz

- Status changed from New to In Progress

#5 - 11/22/2017 07:56 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

#6 - 11/30/2017 07:01 PM - Peter Amstutz

Tom Clegg wrote:

> For the crunch-dispatch-slurm case: Suggest submitting each job with nice=10000, monitoring the actual priority assigned by slurm, and using scontrol to increase the nice value such that the priority is what we want.
>
> [...]

I'm not sure I follow how this is supposed to work. How do we determine the target priority? Is the idea to go and check if there is already a job with the same priority and then fiddle with the niceness so that the actual slurm priority matches the target or gets as close as possible? This seems complex to implement, it requires maintaining additional global state mapping container priority to job priority (which is a moving target).

Proposed alternate solution: calculate niceness as (1000 - p) * 1000. This gives a bias for newer higher priority job to run before an older lower priority job. If we want a stronger bias, we could compress the Arvados priority range to 1-100 and calculate niceness as (100 - p) * 10000.

#8 - 12/01/2017 03:07 PM - Peter Amstutz

Manually tested on 4xphq. Setting initial niceness and changing niceness dynamically work.

#9 - 12/01/2017 05:06 PM - Tom Clegg

Peter Amstutz wrote:

> Tom Clegg wrote:
>
> > For the crunch-dispatch-slurm case: Suggest submitting each job with nice=10000, monitoring the actual priority assigned by slurm, and using scontrol to increase the nice value such that the priority is what we want.

(I should have said "...decrease the nice value")

> I'm not sure I follow how this is supposed to work. How do we determine the target priority? Is the idea to go and check if there is already a job with the same priority and then fiddle with the niceness so that the actual slurm priority matches the target or gets as close as possible? This seems complex to implement, it requires maintaining additional global state mapping container priority to job priority (which is a moving target).
>
> Proposed alternate solution: calculate niceness as (1000 - p) * 1000. This gives a bias for newer higher priority job to run before an older lower priority job. If we want a stronger bias, we could compress the Arvados priority range to 1-100 and calculate niceness as (100 - p) * 10000.

The objective is to have the slurm prioritization order match the arvados prioritization order, regardless of the order the jobs were submitted.

For example, say we submit jobs for containers A, B, C in that order, with the same niceness because they all have priority=100.

- container A → job 1001 → slurm priority 4294000900
- container B → job 1002 → slurm priority 4294000700
- container C → job 1003 → slurm priority 4294000500

Then an Arvados user updates priority A→200, B→100, C→200. If we translate our "priority" directly to a slurm "niceness" offset, we get something like:

- container A → job 1001 → slurm priority 4294001000
- container B → job 1002 → slurm priority 4294000700
- container C → job 1003 → slurm priority 4294000600

Now, relative priority is C>B in Arvados, but C<B in slurm. I don't think this problem can be solved by choosing a scale factor. Slurm adjusts priority to account for submission order, and we need to be able to override that.

OTOH, if we read the relative priority back from slurm, we can add/subtract niceness until the priority order is what we want, i.e., "order by priority desc, created_at":

- container A → job 1001 → slurm priority 4294000900
- container B → job 1002 → slurm priority 4294000700
- container C → job 1003 → slurm priority 4294000500 -- wrong order, need to adjust niceness by -201 to -399

Of course the adjustment is possible only if C was submitted with nice≥201. I'm guessing the races will work out a bit better if we submit with high niceness and reduce, rather than submit with low niceness and increase. But aside from races, either way should work.

Other comments

Using a niceness range of 0..10000 instead of 0..1000000 would make this compatible with slurm 2.6.5 in ubuntu 1404.

Containers API says container_request priority is NULL if and only if state!=Committed. Changing that to "priority is 0 if state!=Committed" sounds fine, but I don't think it should be >0 when state!=Committed.

#10 - 12/01/2017 05:11 PM - Peter Amstutz

You didn't answer the key question:

> How do we determine the target priority? Is the idea to go and check if there is already a job with the same priority and then fiddle with the niceness so that the actual slurm priority matches the target or gets as close as possible? This seems complex to implement, it requires maintaining additional global state mapping container priority to job priority (which is a moving target).

#11 - 12/01/2017 07:08 PM - Peter Amstutz

In your example:

```
container A → job 1001 → niceness 0 → slurm priority 4294000900
container B → job 1002 → niceness 0 → slurm priority 4294000700
container C → job 1003 → niceness 0 → slurm priority 4294000500
```

Then an Arvados user updates priority A→200, B→100, C→200. Using biased niceness (1000 - p) * 1000 this becomes:

```
container A → job 1001 → niceness 800000 → slurm priority 4293200900
container B → job 1002 → niceness 900000 → slurm priority 4293100700
container C → job 1003 → niceness 800000 → slurm priority 4293200500
```

As desired, 4293200900 (A) > 4293200500 (B) > 4293100700 (C)

Your proposal is complicated because it requires choosing anchoring values and constantly re-anchoring as jobs move through the queue:

eg:

```
container A → job 1001 → niceness 0 → slurm priority 4294000002
container B → job 1002 → niceness 0 → slurm priority 4294000001
container C → job 1003 → niceness 0 → slurm priority 4294000000
```

Set niceness directly:

```
container A → job 1001 → niceness 800 → slurm priority 4294999202
container B → job 1002 → niceness 900 → slurm priority 4294999101
container C → job 1003 → niceness 800 → slurm priority 4294999200
```

Now if we want A and C to be the same priority, we need to choose anchoring values. So we say Arvados priority 200 = 4294000000

```
container A → job 1001 → niceness   2 → slurm priority 4294000000
container B → job 1002 → niceness 101 → slurm priority 4293999900
container C → job 1003 → niceness   0 → slurm priority 4294000000
```

So far so good. Now we add container D which also has arvados priority 200.

```
container D → job 1004 → niceness   0 → slurm priority 4293999999
```

Oops, if we want container D to have the same slurm priority, now we need to rebalance everything based on 200 = 4293999999

```
container A → job 1001 → niceness   3 → slurm priority 4293999999
container B → job 1002 → niceness 102 → slurm priority 4293999899
container C → job 1003 → niceness   1 → slurm priority 4293999999
container D → job 1004 → niceness   0 → slurm priority 4293999999
```

We can buy some time if we set the anchor lower, say 1000 below the lowest priority job, say 200 = 4293999000:

```
container A → job 1001 → niceness 802 → slurm priority 4293999000
container B → job 1002 → niceness 901 → slurm priority 4293998900
container C → job 1003 → niceness 800 → slurm priority 4293999000
```

Now when we add container D, we don't have to readjust the other priorities immediately:

```
container D → job 1004 → niceness 799 → slurm priority 4293999999
```

But we can't keep this game up forever, the underlying slurm priority mechanism means every 1000 jobs we'll still need to re-anchor and update the niceness of every job.

My point is that this approach is complicated, counter intuitive, likely hard to get right, and may create scaling problems due to the need to periodically update thousands of jobs. If it is important to set priorities exactly, then we should just set the priority directly (and accept that requires crunch-dispatch-slurm to run as root). If we want to work through the niceness mechanism, instead of trying to be clever, we should use the biasing that is currently implemented.

The biasing doesn't absolutely guarantee that a higher priority job will beat out a lower priority one, on the other hand, it does mean lower priority jobs that "wait their turn" get to run eventually.

If we're going to limit the niceness range to 1-10000 (instead of 1000000) then we should compress the arvados range of 1-100 and make the bias * 100.

#12 - 12/01/2017 07:11 PM - Tom Clegg

Peter Amstutz wrote:

> You didn't answer the key question:
>
> > How do we determine the target priority? Is the idea to go and check if there is already a job with the same priority and then fiddle with the niceness so that the actual slurm priority matches the target or gets as close as possible? This seems complex to implement, it requires maintaining additional global state mapping container priority to job priority (which is a moving target).

Sorry I wasn't clear. There is no target priority. There is only a target priority **order.**

The strategy is to compare the Arvados containers against their corresponding slurm jobs to find out whether the (Arvados-submitted) slurm jobs are in the correct priority order. When they are, we just leave slurm alone. When they're not, we make them so by adjusting niceness. (Except some edge cases on restricted-nice-range slurm versions where it could actually be impossible.)

You could describe the "compare-and-fix" procedure as computing a correct (and achievable) mapping of container UUID to slurm job priority, and then implementing it by calling scontrol update. I'm not sure if this is what you mean by maintaining global state. I don't think it has to be "maintained" in the sense of persisting it between one compare-and-fix operation and the next, though. At any given moment we can call squeue and /arvados/v1/containers and get all the information we need to do a compare-and-fix. And again for clarity, it's a mapping of container **uuid** to slurm job priority, not container **priority** to slurm job priority. If two containers have equal priority, the the one that was submitted first should have a higher slurm priority.

I agree this is not as simple as translating priority to niceness on a fixed scale. I do prefer simple solutions. But on the other hand, it seems like this would actually work, and the simple solution wouldn't, so it's kind of a moot point. Am I missing something?

#13 - 12/01/2017 07:29 PM - Tom Clegg

In the example in note-11 where the simple solution works, it works only because the priority-to-niceness magnification factor (1000) is larger than the distance between slurm-assigned job priorities (100). Then, to accommodate older versions of slurm, we need to reduce the magnification factor to 100, which seems to mean at production scale users would have to guess how much to inflate their priority figures.

I'm also wondering about the problem of multiple workflows submitted with equal priority, which currently results in containers being interleaved and all of the workflows finishing late, where users seem to prefer something more FIFO-like. If we implement priority by adjusting slurm priorities to exactly the desired order, we could address that by using each container's top level container submission time (rather than the submission time of the container itself) as the tie breaker. Does the static magnification approach leave us an opportunity to address this?

#14 - 12/01/2017 08:20 PM - Peter Amstutz

Tom Clegg wrote:

> You could describe the "compare-and-fix" procedure as computing a correct (and achievable) mapping of container UUID to slurm job priority, and then implementing it by calling scontrol update. I'm not sure if this is what you mean by maintaining global state. I don't think it has to be "maintained" in the sense of persisting it between one compare-and-fix operation and the next, though. At any given moment we can call squeue and /arvados/v1/containers and get all the information we need to do a compare-and-fix. And again for clarity, it's a mapping of container **uuid** to slurm job priority, not container **priority** to slurm job priority. If two containers have equal priority, the the one that was submitted first should have a higher slurm priority.

Ok, so going back to my example, we want priority order "A C B":

```
container A → job 1001 → niceness 1000 → slurm priority 4293999002
container B → job 1002 → niceness 1002 → slurm priority 4293998999
container C → job 1003 → niceness 1000 → slurm priority 4293999000
```

Now I create container D, and I want the priority order to be "A C D B". What should its niceness be?

```
container D → job 1004 → niceness 1000 → slurm priority 4293998999 (ripple down: must change priority of B)

container D → job 1004 → niceness  999 → slurm priority 4293999000 (ripple up: must change priority of C)
```

In a pathological case, this could require O(n^2) niceness updates. I suppose we could make it O(n log_2 n) by sorting everything and re-assigning niceness to every job (still requires n niceness updates). Doing ten thounsand invocations of "scontrol" seems... not great.

#15 - 12/01/2017 08:23 PM - Peter Amstutz

Just wanted to point out these cases are not at all contrived, if two workflows are running concurrently at different priority, having jobs pop up in the

middle of the queue (ie a new priority 200 job is below all the other priority 200 jobs but above all the priority 100 jobs) is going to happen all the time.

#16 - 12/01/2017 08:54 PM - Peter Amstutz

Tom Clegg wrote:

> I'm also wondering about the problem of multiple workflows submitted with equal priority, which currently results in containers being interleaved and all of the workflows finishing late, where users seem to prefer something more FIFO-like. If we implement priority by adjusting slurm priorities to exactly the desired order, we could address that by using each container's top level container submission time (rather than the submission time of the container itself) as the tie breaker. Does the static magnification approach leave us an opportunity to address this?

Sort of, my thought was to manage the priority of the overall workflow: https://dev.arvados.org/issues/12552#note-2

It requires something external to keep a priority counter. And of course it would bottom out from time to time. So it's not ideal.

#17 - 12/01/2017 09:17 PM - Peter Amstutz

It's still my opinion that we should go ahead and merge the "simple" niceness strategy (which is still better than nothing) and see how well it works in practice, and then follow up with something more complex that also takes #12552 into account.

#18 - 12/01/2017 09:38 PM - Peter Amstutz

> Containers API says container_request priority is NULL if and only if state!=Committed. Changing that to "priority is 0 if state!=Committed" sounds fine, but I don't think it should be >0 when state!=Committed.

Fixed in 0b1508df7ee4526340e8834422dd49ced63ee8d9 (priority defaults to 0).

#19 - 12/01/2017 09:42 PM - Tom Clegg

I think something like this would behave reasonably well, where

- new jobs are submitted with nice=10000
- config.PriorityElbowRoom is
  - ideally 1000
  - lower (10?) on sites where slurm nice values only go to 10000
  - lower (0?) on sites where Arvados-submitted jobs need to compete with other slurm tenants and it's more important to get priority higher than to minimize scontrol invocations

```
func reprioritize() {
    wantQueue := []struct {
        ctr      *arvados.Container
        priority int // current slurm priority
        nice     int // current slurm nice value
    }{}

    // Get current set of "our" containers (queued, running, and
    // locked) into wantQueue

    // Call "squeue" and fill in "priority" and "nice" fields of
    // wantQueue

    // Sort wantQueue by ctr.Priority desc, ctr.CreatedAt asc (i.e.,
    // wantQueue[0] should end up with the highest slurm priority)

    // Check & fix:
    var nextPriority int
    for i, ent := range wantQueue {
        adjust := 0    // how much priority should increase
        if i == 0 {
            // Might as well make the highest priority job
            // nice=0.
            adjust = ent.nice
        } else if ent.priority > nextPriority {
            // Current slurm priority is too high.
            adjust = nextPriority - ent.priority
            if i == len(wantQueue) || wantQueue[i+1].priority > nextPriority-1 {
                // We're going to renice the next item
                // anyway (or this is the last item).
                // We might as well make some elbow
                // room so we don't go full N^2.
                adjust -= config.PriorityElbowRoom
```

```
            }
            setSlurmNice(ent.ctr.UUID, ent.nice-adjust)
        } else if ent.priority < nextPriority-config.PriorityElbowRoom*10 {
            // Too much distance. This ensures we pull
            // newly submitted jobs up to a reasonable
            // priority number, even if they're at the
            // back of the queue.
            adjust = nextPriority - ent.priority - config.PriorityElbowRoom
        }
        if adjust > ent.nice {
            // Can't adjust that much without being root.
            // Just go as far as we can.
            adjust = ent.nice
        }
        if adjust != 0 {
            setSlurmNice(ent.ctr.UUID, ent.nice-adjust)
        }
        nextPriority = ent.priority + adjust - 1
    }
}

func setSlurmNice(ctrUUID string, newNiceValue int) {
    // ...
}
```

#20 - 12/04/2017 06:35 PM - Peter Amstutz

- Related to Story #12552: When priority is equal, the children of an earlier-submitted workflow should run first added

#21 - 12/04/2017 06:45 PM - Peter Amstutz

My understanding of engineering discussion:

Go to go with simple priority->niceness (1000 - priority) * 10 for now with guidelines on how to use priority to achieve the desired behavior, revisit other approaches (such as #note-19) when grooming #12552.

Updated documentation to reflect interpretation of priority implemented on this branch.

eaf301421b5a71e8344688723c3852e7bd5154b8

#22 - 12/04/2017 07:39 PM - Tom Clegg

Yes, it seems like the simple approach (pass 0-priority as slurm "nice") does address the problem as stated: priority will no longer be ignored. We can defer the more predictable behavior to #12552.

Docs

I don't think the docs should imply that the interaction between priority and submission time/order is intentional. I would say that *in the current implementation,* the relative priority of two jobs is also affected by the number of other slurm jobs submitted between them; in practice, a larger priority interval is more likely to affect actual execution order.

I don't think Container docs should promise container priority == max(container request priority). Inserting the word "currently" would work. (I imagine we'll want different users/projects to have different priority weight once everyone learns to submit everything with priority=1000.)

The "_container_scheduling_parameters" fragment is included on both container_request and container pages. The "no container should run on behalf of this request" paragraph is a bit of a non sequitur on the container page. Maybe this added section is only needed on the container request page, since that's generally the thing users work with?

Code

This part of set_requesting_container_uuid() looks like part of #12574, is that right?

```
+    if container
+      self.requesting_container_uuid = container.uuid
+      self.priority = container.priority
+    end
```

This seems mysterious:

```
     cr = create_minimal_req!(priority: nil)
+    cr.priority = nil
```

While we're adding validations, why is this allowed? Shouldn't priority=1 be rejected when state=Uncommitted?

```
+       cr.priority = 0
+       cr.save!
+
+       cr.priority = 1
+       cr.save!
```

The "Container request valid priority" test in ContainerTest looks like it's really "Container valid priority" and all of its "cr" should change to "c".

#23 - 12/04/2017 08:12 PM - Peter Amstutz

Tom Clegg wrote:

> Yes, it seems like the simple approach (pass 0-priority as slurm "nice") does address the problem as stated: priority will no longer be ignored. We can defer the more predictable behavior to #12552.
>
> ### Docs
>
> I don't think the docs should imply that the interaction between priority and submission time/order is intentional. I would say that *in the current implementation,* the relative priority of two jobs is also affected by the number of other slurm jobs submitted between them; in practice, a larger priority interval is more likely to affect actual execution order.

Done.

> I don't think Container docs should promise container priority == max(container request priority). Inserting the word "currently" would work. (I imagine we'll want different users/projects to have different priority weight once everyone learns to submit everything with priority=1000.)

Done.

> The "_container_scheduling_parameters" fragment is included on both container_request and container pages. The "no container should run on behalf of this request" paragraph is a bit of a non sequitur on the container page. Maybe this added section is only needed on the container request page, since that's generally the thing users work with?

Done.

> ### Code
>
> This part of set_requesting_container_uuid() looks like part of #12574, is that right?
> [...]

Correct. But I missed it in #12574 so I put it in this branch instead.

> This seems mysterious:
> [...]

Yes, that is confusing, isn't it. Took out the meaningless initializer.

> While we're adding validations, why is this allowed? Shouldn't priority=1 be rejected when state=Uncommitted?
> [...]

If I'm understanding you correctly, rejecting it that would prevent you from setting the priority in one API call and then committing the request in a separate call. That might break workbench (which creates the container request for editing & then commits it separately.)

> The "Container request valid priority" test in ContainerTest looks like it's really "Container valid priority" and all of its "cr" should change to "c".

Sorry, sloppy cut and paste. Fixed.

Now @ d7b909ceb2646ef0637daaea70c768b799670c60

#24 - 12/04/2017 09:18 PM - Tom Clegg

Peter Amstutz wrote:

> If I'm understanding you correctly, rejecting it that would prevent you from setting the priority in one API call and then committing the request in a separate call. That might break workbench (which creates the container request for editing & then commits it separately.)

Yes, I was thinking we could forbid {state=Uncommitted, priority>0} for being meaningless. But we didn't come here to make validation more strict, did we. Happy to leave it alone.

The docs suggest queue time competes with priority. I thought it was number of intervening slurm jobs, not time -- but whatever, I think you've made the main point, i.e., priority+=1 probably won't do much.

LGTM, thanks

#25 - 12/04/2017 09:40 PM - Anonymous

- Status changed from In Progress to Resolved

- % Done changed from 50 to 100

Applied in changeset arvados|bcf069c6a726e219bc40224653268d87776e54aa.