Pulp - Issue #4296

Stages API could deadlock when "discovering" content due to minsize

01/04/2019 10:12 PM - bmbouter

Status:	CLOSED - CURRENTRELEASE	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	mdellweg	**Estimated time:**	0:00 hour
Category:			
Sprint/Milestone:	3.0.0		
Severity:	2. Medium	**Groomed:**	No
Version:		**Sprint Candidate:**	No
Platform Release:		**Tags:**	
OS:		**Sprint:**	
Triaged:	Yes	**Quarter:**	

Description

This was originally described by mdellweg in this comment. What he is describing is a pipeline where content is "discovered" as it goes, similar to what issue 4294 will support.

Generally there are situations where the pipeline is holding content in a batch which is needed to in-turn fill up the batch to minsize.

The Solution

We add a field `does_batch` to the `DeclarativeContent` that defaults to True.
This value can be overwritten by an argument to `__init__`, and it gets cleared, when a future is requested.
When does_block is false, the batching mechanism does not wait for any further content, that is not immediately available, to prevent blocking and thereby deadlocking.

Related issues:

Blocks Container Support - Refactor #4173: Change the multilayered design to ...	**CLOSED - CURRENTRELEASE**

History

#1 - 01/04/2019 10:21 PM - bmbouter

- Subject changed from Stages API could deadlock when "discovering" content to Stages API could deadlock when "discovering" content due to minsize

- Description updated

#2 - 01/04/2019 10:31 PM - bmbouter

mdellweg had an idea that the DeclarativeContent could be marked somehow for it to not be batched. This solves the issue by disabling batching without having to pass options to all of the stages.

Another option is to introduce a timer to Stage.batches. Overall I think that would be an important safety feature to avoiding deadlock. For example we could add a max_timer=1000 where max_timer is the maximum number of milliseconds to hold > 0 items before returning the batch even if minsize is not met. Maybe plugin writers would need to override this option also?

#3 - 01/07/2019 10:28 AM - mdellweg

In my last version of a proposed solution, i used the tagged DeclarativeContent objects to stop waiting for the current batch to fill. So the would not circumvent the batching completely, just cutting them shorter.

And there is another solution i can think of:
We could introduce, like `None` as the pipeline ends marker, a `flush` marker that thaws all pending batches on the way, but not finish the loop. This would, however, require that the DCs do not reorder from step to step.

#4 - 01/08/2019 04:44 PM - bmbouter

mdellweg wrote:

> In my last version of a proposed solution, i used the tagged DeclarativeContent objects to stop waiting for the current batch to fill. So the would not circumvent the batching completely, just cutting them shorter.

I imagined tagging all DeclarativeContent, but you're identifying you would just tag some of them. Can you describe that more? How do you know which DeclarativeContent items could cause deadlock?

> And there is another solution i can think of:
> We could introduce, like `None` as the pipeline ends marker, a `flush` marker that thaws all pending batches on the way, but not finish the loop.
> This would, however, require that the DCs do not reorder from step to step.

We have some stages that do reordering now so I already have that assumption as a property of the Stages API.

#5 - 01/08/2019 04:51 PM - CodeHeeler

- Triaged changed from No to Yes

#6 - 01/11/2019 12:38 PM - mdellweg

bmbouter wrote:

> I imagined tagging all DeclarativeContent, but you're identifying you would just tag some of them. Can you describe that more? How do you know which DeclarativeContent items could cause deadlock?

I think it is as simple as "Can this content unit give raise to more content units? Then make it prioritized."
In the case of Debian Repositories, the discovery chain has 3 Levels: Release -> PackageIndex -> Package. So I give the priority Flag to all but the last Level. Since the hugely overwhelming part of content units are leafs in this picture, this i not a big performance concern.

If you only have one content type, and you never know in advance, whether it can produce more, you are out of luck here. Then I could think of a kind of barrier (flush marker), that gets inserted whenever the first stage has nothing more to add, and you must make sure, no content unit get's overtaken by that barrier (It might help performance a little if the barrier was semipermeable).

#7 - 01/11/2019 09:11 PM - amacdona@redhat.com

- Blocks Refactor #4173: Change the multilayered design to use Futures to handle nested content added

#8 - 01/14/2019 11:58 AM - mdellweg

Before changing the flow control of the pipeline, i would like to change the plugin api in a way, that the abstracts the interface to the queues away from plugin writers.

RFC: https://github.com/mdellweg/pulpcore-plugin/tree/refactor_stages

#9 - 01/30/2019 02:25 PM - bherring

- Related to Test #4359: 2.18.1 Testing added

#10 - 01/30/2019 02:26 PM - bherring

- Related to deleted (Test #4359: 2.18.1 Testing)

#11 - 02/06/2019 10:16 PM - bmbouter

- Status changed from NEW to POST

- Assignee set to mdellweg

PR available at: https://github.com/pulp/pulpcore-plugin/pull/48

#12 - 02/08/2019 04:39 PM - mdellweg

- Description updated

#13 - 02/08/2019 06:39 PM - mdellweg

- Status changed from POST to MODIFIED

Applied in changeset commit:pulpcore-plugin|42f17e72a0b399c7eb6c61e9722da12021a278e6.

#14 - 04/25/2019 06:44 PM - daviddavis

- Sprint/Milestone set to 3.0.0

#15 - 04/26/2019 10:32 PM - bmbouter

- Tags deleted (Pulp 3)

#16 - 12/13/2019 06:10 PM - bmbouter

- Status changed from MODIFIED to CLOSED - CURRENTRELEASE