**Observation**

Jobs with `RETRY=...` setting are automatically restarted in case of a failure but apparently the dependency handling is not done in accordance with the normal restart behavior (you would get when e.g. clicking on the restart button in the web UI).

For instance, here the root job has been restarted multiple times but none of the children have been restarted: https://openqa.suse.de/tests/8656146#dependencies

This also leads to a not so nice graph where the multiple clones of the root job are present at the same time:
Acceptance criteria

- AC1: Jobs are restarted in a more sensible way¹ regarding dependencies. Likely there's not one best way but the default should at least work better in most cases.
- AC2: Potential concurrency issues which might be the culprit (or at least contribute to the overall problem) here are investigated and dealt with if needed (see #110458#note-4 for further details).

¹ What "more sensible" means exactly we have still have to define for each dependency type. Maybe it makes most sense to go with the behavior the restart API has by default.

Further ideas

- Allow the user to specify the retry behavior, similar to how it is already possible with the different parameters the restart API supports.

Related issues:
Related to openQA Project - action #112256: Some children of parent job not c... New 2022-06-09

History

#1 - 2022-04-29 08:25 - okurz
- Subject changed from Improve `RETRY=...`-behavior for jobs with dependencies to [epic] Improve `RETRY=...`-behavior for jobs with dependencies
- Priority changed from Normal to Low
- Target version set to future

#2 - 2022-07-05 09:56 - mkittler
- Related to action #112256: Some children of parent job not cancelled (or later, restarted) when parent 'parallel_failed' due to another child's parallel job failing added

#3 - 2022-08-12 10:30 - okurz
- Parent task changed from #103962 to #112862

Move future ideas to the actual "Future ideas" tracker #112862

#4 - 2022-08-29 13:00 - mkittler

The symptoms of this problem can be hard to pin down but I suppose this is another example of this issue: https://suse.slack.com/archives/C02CANHLANP/p1661763124213709

I suppose one problem is that auto_duplicate is invoked around the same time for multiple incompletes of the same parallel cluster.

After having a closer look at the code, I'm not sure whether it is buggy. Here are my thoughts:

- The code for creating clones is using a transaction but not using an isolation level that would prevent it from seeing changes committed by other transactions. Therefore I'm not sure whether the code handles this situation well.
  - At least the upfront check whether there's already an existing clone can definitely not work in all situations.
  - So the algorithm for determining the cluster jobs might run at some point in the middle on a cluster that has already been restarted.
  - The same counts for the code that actually creates the clones and inserts dependencies.
- However, the code does some "optimistic" locking. So in case a job we want to clone "suddenly" already has a clone we abort the transaction. I'm not sure whether this kind of locking is enough, though. E.g. what would happen if:
  1. Process A creates a bunch of closes but hasn't committed the changes yet.
  2. Process B creates the same clones. Since A hasn't committed its changes yet, B will not be prevented to do so as our optimistic locking code will not even see changes from A.
  3. One of the processes commits first. I suppose that should succeed.
  4. The other process commits the changes as well. I'm not sure how it'll be handled by PostgreSQL. Either:
     - It fails discarding all changes. That would be good for us because then it'll be like we only attempted to create the clones once which is what we want.
     - It succeeds, partially overriding changes from the faster transaction. That would leave jobs from the faster transaction dangling. They'd not be considered clones of anything anymore (like the jobs we found today on OSD). And besides, we'd have surplus sets of jobs. Maybe the dependency creation is screwed with as well (like in the jobs we found today on OSD).

It would be interesting to check which of the options in the last point it is.

Note that looking at the timestamps supports the theory that the job cloning was done concurrently because the creation times of the jobs from the relevant "sets restarted jobs" are interleaving:
openqa=# select id, clone_id, (select id from jobs as j2 where clone_id = jobs.id) as cloned_as_id, (select count(*) from job_dependencies where parent_job_id = jobs.id or child_job_id = jobs.id) as direct_dependency_count, t_created, result from jobs where id in (9399731, 9401905, 9401919, 9401927, 9401918, 9401926, 9401944, 9401969, 9401980) order by t_created;

<table>
<thead>
<tr>
<th>id</th>
<th>clone_id</th>
<th>cloned_as_id</th>
<th>direct_dependency_count</th>
<th>t_created</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>9399731</td>
<td>9401905</td>
<td></td>
<td></td>
<td>2022-08-27 20:33:32</td>
<td>incomplete</td>
</tr>
<tr>
<td>9401905</td>
<td></td>
<td>9399731</td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>passed</td>
</tr>
<tr>
<td>9401918</td>
<td></td>
<td></td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>timeout_exceeded</td>
</tr>
<tr>
<td>9401919</td>
<td></td>
<td>9399733</td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>passed</td>
</tr>
<tr>
<td>9401926</td>
<td></td>
<td></td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>timeout_exceeded</td>
</tr>
<tr>
<td>9401927</td>
<td></td>
<td>9399734</td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>passed</td>
</tr>
<tr>
<td>9401944</td>
<td>9402855</td>
<td></td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>timeout_exceeded</td>
</tr>
<tr>
<td>9401946</td>
<td></td>
<td>9399733</td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>passed</td>
</tr>
<tr>
<td>9401969</td>
<td>9402856</td>
<td></td>
<td></td>
<td>2022-08-28 01:36:36</td>
<td>timeout_exceeded</td>
</tr>
<tr>
<td>9401980</td>
<td>9402857</td>
<td></td>
<td></td>
<td>2022-08-28 01:36:40</td>
<td>timeout_exceeded</td>
</tr>
</tbody>
</table>

Note that if concurrent cloning/restarting is indeed not handled well that means this bug is actually not really specific to the RETRY=...-feature. It would theoretically also happen if a user manages to click the restart buttons of multiple jobs in the cluster simultaneously but that's of course very unlikely to happen.

- Description updated

#5 - 2022-08-29 13:03 - mkittler

The code for creating clones is using a transaction but not using an isolation level that would prevent it from seeing changes committed by other transactions. Therefore I'm not sure whether the code handles this situation well.

- At least the upfront check whether there's already an existing clone can definitely not work in all situations.
- So the algorithm for determining the cluster jobs might run at some point in the middle on a cluster that has already been restarted.
- The same counts for the code that actually creates the clones and inserts dependencies.

However, the code does some "optimistic" locking. So in case a job we want to clone "suddenly" already has a clone we abort the transaction. I'm not sure whether this kind of locking is enough, though. E.g. what would happen if:

1. Process A creates a bunch of closes but hasn't committed the changes yet.
2. Process B creates the same clones. Since A hasn't committed its changes yet, B will not be prevented to do so as our optimistic locking code will not even see changes from A.
3. One of the processes commits first. I suppose that should succeed.
4. The other process commits the changes as well. I'm not sure how it'll be handled by PostgreSQL. Either:
   - It fails discarding all changes. That would be good for us because then it'll be like we only attempted to create the clones once which is what we want.
   - It succeeds, partially overriding changes from the faster transaction. That would leave jobs from the faster transaction dangling. They'd not be considered clones of anything anymore (like the jobs we found today on OSD). And besides, we'd have surplus sets of jobs. Maybe the dependency creation is screwed with as well (like in the jobs we found today on OSD).

I've not looked at the code, but to me it sounds like both transactions should succeed and create clones, unless the data from the second transaction actually conflicts with the data from the first transaction (unique constraints on the table, or similar). And if there's a conflict the thrown exception should result in a full rollback of the whole transaction, which doesn't appear to be the case here. Perhaps a SELECT FOR UPDATE would be useful here to make sure the second transaction gets blocked early enough by the first one.

#9 - 2022-08-29 13:40 - mkittler

Thanks. Maybe SELECT FOR UPDATE would be helpful, indeed. It is even supported by DBIx::Class (see [https://metacpan.org/pod/DBIx::Class::ResultSet#force](https://metacpan.org/pod/DBIx::Class::ResultSet#force)).

Files