Arvados - Feature #3448

[Keep] storage server records (and reports in index) the timestamp of the most recent PUT operation for each block.

07/31/2014 09:15 PM - Tom Clegg

Status:	Resolved	**Start date:**	08/21/2014
Priority:	Normal	**Due date:**	
Assigned To:	Tim Pierce	**% Done:**	100%
Category:	Keep	**Estimated time:**	0.00 hour
Target version:	2014-08-27 Sprint		

Description

This statistic is needed to ensure a grace period (equal to permission signature TTL) after writing a block, during which the block is not eligible for garbage collection.

- Data manager needs to know this timestamp (via storage server's GET / index method) in order to get a more accurate picture of which blocks it will be able to delete.
- Storage server itself needs to check this timestamp when performing DELETE requests, to catch race conditions (if a client PUT a new copy of a block 10 seconds ago, after data manager decided 20 seconds ago that the block was a good candidate for deletion, the storage server should refuse to delete it).

Implementation

- Use mtime (touch(1), godoc syscall → syscall.Utimes()?) to store timestamps. This makes it easy for an admin to inspect state using the usual filesystem tools.
  - Volume and UnixVolume should grow an UpdateTimestamp method.
  - GetBlock() in handlers.go should accept an additional argument indicating whether it should also perform an "update timestamp" operation before considering its "Get" operation successful.
- Readonly cases: timestamps can't get updated, so clients won't get a grace period. The safest option is to err on the side of excessive replication: when PUTting a block that already exists on the volume, if the timestamp cannot be updated, consider that a failure and try the next volume.
- In DELETE, check the mtime of each file before deleting it. If it's newer than permission signature TTL, don't delete. There's a race condition here too, so use mutexes around {check timestamp; delete file} and around {update timestamp during PUT}. If there's a race, "update timestamp" will fail ENOENT and the PUT operation should either retry the current volume once (which would involve writing) or just give up and move on to the next volume.
  - Wrap each of the {check timestamp; delete file} and {update timestamp} operations with {open file with mode "a+" and Flock(LOCK_EX) it}
  - (Considered using an in-process lock like the -serialize-io mechanism, but that doesn't guard against accidental misconfigurations where multiple keepstore processes point at the same mount point.)
- UnixVolume's Index method already reports modtime. Yay!

Subtasks:

Task # 3647: Add timestamp support to PUT and DELETE	**Resolved**
Task # 3683: Review 3448-keep-put-timestamps	**Resolved**

Associated revisions

Revision 56d36387 - 08/21/2014 01:55 PM - Tim Pierce

3448: check block timestamp before DELETE

volume.Delete locks the target file and checks the timestamp before proceeding. If the file is newer than permission_ttl specifies, return a PermissionError. This way, a block that has been marked for recycling by Data Manager but subsequently was re-added by a user will not be prematurely deleted.

PutBlock now updates the timestamp on the target block if it already exists on disk, to prevent DELETE from recycling old blocks that have just been refreshed.

To update block timestamps, the blob server uses the new volume.Touch method. MockVolume and UnixVolume have been updated appropriately.

Refs #3448.

Revision 12e80e52 - 08/21/2014 02:25 PM - Tim Pierce

3448: minor bugfixes

Refs #3448.

Revision dc2bf5c8 - 08/21/2014 03:55 PM - Tim Pierce

3448: code review comments.

Extend GetBlock() to optionally update the file modification time, so PUT operations can update the timestamp of an existing block.

UnixVolume.Delete() returns nil if the file is too new to delete (the reasoning here is that this is the correct thing for the server to do, even if the result technically does not fulfill the user's request, so the server should return success).

Refs #3448.

Revision 1080ec93 - 08/21/2014 04:33 PM - Tim Pierce

3448: unit tests for deleting new blocks

Added cases to TestDeleteHandler to test that blocks newer than -permission_ttl will not be removed from the volume even if volume.Delete() returned true.

Refs #3448.

Revision 3616a539 - 08/21/2014 04:43 PM - Tim Pierce

3448: add error checking

Code review comments in https://arvados.org/issues/3448#note-9

- volume.Touch() must fail if the file cannot be locked
- if volume.Touch() fails, continue looking for other volumes that can fulfill the request.

Refs #3448.

Revision e386d206 - 08/21/2014 07:00 PM - Tim Pierce

3448: add error checking in volume.Touch()

Add error checking that was erroneously left out of previous commit.

Refs #3448.

Revision 94b5a596 - 08/25/2014 01:31 PM - Tim Pierce

Merge branch '3448-keep-put-timestamps'

Closes #3448.

History

#1 - 07/31/2014 09:37 PM - Tom Clegg

- Description updated

- Category set to Keep

#2 - 07/31/2014 09:37 PM - Tom Clegg

- Story points changed from 1.0 to 2.0

#3 - 08/06/2014 04:12 PM - Tim Pierce

- Assigned To set to Tim Pierce

#4 - 08/19/2014 02:55 PM - Tom Clegg

- Description updated

#5 - 08/21/2014 02:11 PM - Tim Pierce

- Status changed from New to In Progress

#6 - 08/21/2014 02:39 PM - Tom Clegg

At 56d36387

In handlers.go, we call GetBlock(), which walks through some/all of the volumes looking for the block, then call TouchBlock(), which walks through the same volumes looking for the same block, in order to update its timestamp. Aside from the wasted effort, this makes me less confident that the timestamp that gets updated is the one on the very same file whose content we took the trouble to verify. So I still think this is the way to go:

- "GetBlock() in handlers.go should accept an additional argument indicating whether it should also perform an "update timestamp" operation before considering its "Get" operation successful."

I think PermissionError is the wrong response for "race condition detected and handled correctly". 202 ("Accepted") isn't perfect, but it's closer. Even 200 would be reasonable, given the outcome is indistinguishable from "delete succeeded, **then** client wrote anew". (This doesn't need too much bikeshedding given that we're soon moving to an asynchronous "ok-to-delete list" approach.)

The comment in GetBlock() in handlers.go still leaves open the mystery of why we want anything to be writable. Propose: "When reporting success for a PUT operation, we are assuring the caller that the block will not be garbage-collected sooner than permission_ttl. In order to promise this, we must ensure that the timestamp is current, and that all race conditions are accounted for."

#7 - 08/21/2014 03:34 PM - Tim Pierce

Ready for another look: dc2bf5c

Tom Clegg wrote:

> At 56d36387
>
> In handlers.go, we call GetBlock(), which walks through some/all of the volumes looking for the block, then call TouchBlock(), which walks through the same volumes looking for the same block, in order to update its timestamp. Aside from the wasted effort, this makes me less confident that the timestamp that gets updated is the one on the very same file whose content we took the trouble to verify. So I still think this is the way to go:
>
> - "GetBlock() in handlers.go should accept an additional argument indicating whether it should also perform an "update timestamp" operation before considering its "Get" operation successful."

OK, I see why that makes more sense. The interfaces here are getting more and more tangled; I'll file a story with some cleanup ideas that we can revisit after 1.0.

> I think PermissionError is the wrong response for "race condition detected and handled correctly". 202 ("Accepted") isn't perfect, but it's closer. Even 200 would be reasonable, given the outcome is indistinguishable from "delete succeeded, **then** client wrote anew". (This doesn't need too much bikeshedding given that we're soon moving to an asynchronous "ok-to-delete list" approach.)

Fair enough. If the model here isn't "I'm returning an error because I couldn't do the thing you asked me to do" but "I did the right thing for this request based on conditions, even though the file wasn't deleted" then that makes sense. Returning success is by far the easiest thing to do here, but we can tease it into delivering 202 or something.

> The comment in GetBlock() in handlers.go still leaves open the mystery of why we want anything to be writable. Propose: "When reporting success for a PUT operation, we are assuring the caller that the block will not be garbage-collected sooner than permission_ttl. In order to promise this, we must ensure that the timestamp is current, and that all race conditions are accounted for."

That's fine, but I don't understand which comment needs clarifying. (Did you mean this comment on Delete()? https://arvados.org/projects/arvados/repository/revisions/56d363875a42c1abd485c67e030f625299548aa0/entry/services/keepstore/volume_unix.go#L254)

#8 - 08/21/2014 04:15 PM - Tom Clegg

Tim Pierce wrote:

> - "GetBlock() in handlers.go should accept an additional argument indicating whether it should also perform an "update timestamp" operation before considering its "Get" operation successful."
>
> OK, I see why that makes more sense. The interfaces here are getting more and more tangled; I'll file a story with some cleanup ideas that we can revisit after 1.0.

Hm, I recommend you don't invest toooo much time in that until we've incorporated (or at least planned) a few more of the pre-1.0 features.

> Fair enough. If the model here isn't "I'm returning an error because I couldn't do the thing you asked me to do" but "I did the right thing for this

> request based on conditions, even though the file wasn't deleted" then that makes sense. Returning success is by far the easiest thing to do here, but we can tease it into delivering 202 or something.

200 is probably best. 202 isn't quite right, and really, everything is 100% OK when this race happens.

> > The comment in GetBlock() in handlers.go still leaves open the mystery of why we want anything to be writable. Propose: "When reporting success for a PUT operation, we are assuring the caller that the block will not be garbage-collected sooner than permission_ttl. In order to promise this, we must ensure that the timestamp is current, and that all race conditions are accounted for."
>
> That's fine, but I don't understand which comment needs clarifying. (Did you mean this comment on Delete()? https://arvados.org/projects/arvados/repository/revisions/56d363875a42c1abd485c67e030f625299548aa0/entry/services/keepstore/volume_unix.go#L254)

Sorry, I got the file right (handlers.go) but it's in PutBlock, not GetBlock. L498 here, about writable replicas:

https://arvados.org/projects/arvados/repository/revisions/56d363875a42c1abd485c67e030f625299548aa0/diff/

Further to that point, I think it's more correct to say that TouchBlock fails if it cannot assure a current timestamp for any reason. All this talk of read-only volumes is just wild speculation about conditions that could cause "assure a current timestamp" to fail...

#9 - 08/21/2014 04:32 PM - Tom Clegg

Looking at dc2bf5c

This looks much closer, thanks.

Noticed a couple of things

- Need error checking on call to lockfile() (if Flock() fails, Touch() must fail)
- Need error checking on call to Touch() (if Touch() fails, PutBlock must keep looking in other volumes, not proclaim success)

#10 - 08/21/2014 04:46 PM - Tim Pierce

Updated at e386d20

Tom Clegg wrote:

> Looking at dc2bf5c
>
> This looks much closer, thanks.
>
> Noticed a couple of things
>
> - Need error checking on call to lockfile() (if Flock() fails, Touch() must fail)
> - Need error checking on call to Touch() (if Touch() fails, PutBlock must keep looking in other volumes, not proclaim success)

Updated with error checking. Also added a rudimentary unit test case to TestDeleteHandler.

#11 - 08/22/2014 11:06 AM - Tom Clegg

Tim Pierce wrote:

> Updated with error checking. Also added a rudimentary unit test case to TestDeleteHandler.

Looks good, thanks.

I think func (v *UnixVolume) Delete in volume_unix.go is still missing its error checking for lockfile(), though.

I feel like we should have two more basic tests:

- PUT (when data is already present) updates the timestamp on the existing file.
- PUT (when data is present but timestamp cannot be updated) writes a second copy.

Are both of those easy enough to write?

(BTW, in commit messages, our system seems to have settled on saying "refs/closes #3448" only in merges/commits to master -- the "3448:" prefix is fine for branch commits.)

#12 - 08/22/2014 01:57 PM - Tim Pierce

at c15f086:

Tom Clegg wrote:

> Tim Pierce wrote:
>
> > Updated with error checking. Also added a rudimentary unit test case to TestDeleteHandler.
>
> Looks good, thanks.
>
> I think func (v *UnixVolume) Delete in volume_unix.go is still missing its error checking for lockfile(), though.

Ugh, yes, I left it out. Committed now (for sure!)

> I feel like we should have two more basic tests:
>
> - PUT (when data is already present) updates the timestamp on the existing file.
> - PUT (when data is present but timestamp cannot be updated) writes a second copy.
>
> Are both of those easy enough to write?

Yes, those are good ideas. Added TestPutTouch and TestPutTouchFails to exercise these conditions.

> (BTW, in commit messages, our system seems to have settled on saying "refs/closes #3448" only in merges/commits to master -- the "3448:" prefix is fine for branch commits.)

I tend to include "refs #xxxx" just in case and so I don't have to try rewriting my history, but I'll try to remember :-)

#13 - 08/22/2014 05:33 PM - Tom Clegg

Tim Pierce wrote:

> I think func (v *UnixVolume) Delete in volume_unix.go is still missing its error checking for lockfile(), though.
>
> Ugh, yes, I left it out. Committed now (for sure!)

Ah yes, there it is.

(I notice you used "e" instead of "err" here. Is there a reason for that? I've become accustomed to seeing "err" all over the place in Go, which seems like a nice convention, so I'm wondering if there is some other style/convention I should be filing this "e" under...)

> I feel like we should have two more basic tests:
>
> - PUT (when data is already present) updates the timestamp on the existing file.
> - PUT (when data is present but timestamp cannot be updated) writes a second copy.
>
> Are both of those easy enough to write?
>
> Yes, those are good ideas. Added TestPutTouch and TestPutTouchFails to exercise these conditions.

In the TestPutBlockTouchFails test, I suspect the "bad" check in Get() will prevent us from ever reaching the code that calls Touch(): L432 of handlers.go will get an error from Get(), look on other volumes, and eventually return, without ever calling Touch(). I think this could be fixed by having an "Untouchable" flag distinct from the ambiguous "Bad" flag. (Or possibly by telling me I'm confused about something.)

I wonder if it would be just as easy for TestPutTouch to set the Mtime to {now minus 5 seconds}, instead of using sleep(1). It seems a shame to make the test suite runtime grow from 0.022s to 1.023s over this. (Even though our Workbench tests take >5 minutes.)

> (BTW, in commit messages, our system seems to have settled on saying "refs/closes #3448" only in merges/commits to master -- the "3448:" prefix is fine for branch commits.)
>
> I tend to include "refs #xxxx" just in case and so I don't have to try rewriting my history, but I'll try to remember :-)

Sure, it's not worth rebasing these commits or anything, I was just cultivating future consistency. (Hm, come to think of it, leaving "refs #xxxx" off the branch commits also helps by preventing you from accidentally FF-merging into master...)

Thanks.

#14 - 08/22/2014 06:49 PM - Tom Clegg

Noticed this seems odd in the test case that returns 200 despite not deleting any data. Shouldn't we expect copies_deleted=0, copies_failed=0?

```
        // Expect response {"copies_deleted":1,"copies_failed":0}
        expected_dc = deletecounter{1, 0}
```

#15 - 08/25/2014 11:00 AM - Tim Pierce

Updates at cca1529:

Tom Clegg wrote:

> (I notice you used "e" instead of "err" here. Is there a reason for that? I've become accustomed to seeing "err" all over the place in Go, which seems like a nice convention, so I'm wondering if there is some other style/convention I should be filing this "e" under...)

There isn't a specific reason. The Go library code and examples tend to use both "err" and "e" for short error checking tests like this. I can revise this to use "err" throughout if it's confusing.

> In the TestPutBlockTouchFails test, I suspect the "bad" check in Get() will prevent us from ever reaching the code that calls Touch(): L432 of handlers.go will get an error from Get(), look on other volumes, and eventually return, without ever calling Touch(). I think this could be fixed by having an "Untouchable" flag distinct from the ambiguous "Bad" flag. (Or possibly by telling me I'm confused about something.)

Very good point. Added a Touchable flag to MockVolume and updated the TODO with some thoughts about clarifying the use of these flags so we don't overload flags like Bad too much.

> I wonder if it would be just as easy for TestPutTouch to set the Mtime to {now minus 5 seconds}, instead of using sleep(1). It seems a shame to make the test suite runtime grow from 0.022s to 1.023s over this. (Even though our Workbench tests take >5 minutes.)

That would be a nice improvement, I just don't want to boil the ocean too much on this right now. Added a TODO.

> Noticed this seems odd in the test case that returns 200 despite not deleting any data. Shouldn't we expect copies_deleted=0, copies_failed=0?

Good point. I'll file a bugfix ticket for this. It'll require redefining the Volume interface a little.

This is tricky because we've defined a kind of three-way logic for success and failure: "success, I deleted the block," "failure, I couldn't delete the block because something was wrong", and.... "success, I could have deleted the block but decided that wasn't the right thing to do." This is why I wanted Keepstore to respond 4xx if a DELETE failed for policy reasons like this -- it allows Keepstore to define a very clear success/failure status, and requires the Data Manager to be responsible for interpreting whether that's a problem or not.

#16 - 08/25/2014 01:27 PM - Tom Clegg

LGTM, please merge.

Tim Pierce wrote:

> > Noticed this seems odd in the test case that returns 200 despite not deleting any data. Shouldn't we expect copies_deleted=0, copies_failed=0?
>
> Good point. I'll file a bugfix ticket for this. It'll require redefining the Volume interface a little.

Meh, skip it. The "delete" API is about to change to "here's a delete list", so the idea of reporting the result of an individual delete operation with an HTTP status code will be moot.

Thanks!

#17 - 08/25/2014 01:35 PM - Tim Pierce

- Status changed from In Progress to Resolved

Applied in changeset arvados|commit:94b5a59631f22e4e57561a6244c24b93db77f589.