Arvados - Bug #12447

crunch-run memory leak

10/12/2017 03:56 PM - Peter Amstutz

Status:	Resolved	**Start date:**	10/13/2017
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2017-10-25 Sprint		

Description

Crunch-run loading a 2 GiB Docker image uses 1.5 GiB of RAM, which is enough on a 3.5 GiB node to prevent fork/exec due to memory pressure.

See https://dev.arvados.org/issues/12433#note-3 and note-4

Subtasks:

Task # 12449: Review 12447-crunch-run-leak	**Resolved**

Related issues:

Related to Arvados - Bug #11583: [crunch-run] Fix excessive memory use	**Resolved**

Associated revisions

Revision 15b5b59f - 10/16/2017 06:01 PM - Peter Amstutz

Merge branch '12447-crunch-run-leak' closes #12447

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

History

#1 - 10/12/2017 04:00 PM - Peter Amstutz

- Description updated

#2 - 10/12/2017 09:01 PM - Peter Amstutz

The memory profiling reports 416MB consistently used for loading the Docker image, this seems to be independent of image size (it seems to be the same for 380 MB or 3.8 GB images).

```
root@4afaf1b2b92b:/tmp# go tool pprof -tree mem1507841414.pprof
480MB of 480.50MB total (99.90%)
Dropped 25 nodes (cum <= 2.40MB)
----------------------------------------------------------+-------------
      flat  flat%   sum%        cum   cum%   calls calls% + context
----------------------------------------------------------+-------------
                                          416MB   100% |   bytes.(*Buffer).ReadFrom
     416MB 86.58% 86.58%      416MB 86.58%                | bytes.makeSlice
----------------------------------------------------------+-------------
                                           64MB   100% |   main.(*CollectionFileWriter).Write
      64MB 13.32% 99.90%       64MB 13.32%                | main.(*CollectionFileWriter).ReadFrom
----------------------------------------------------------+-------------
                                          416MB   100% |   io/ioutil.readAll
         0     0% 99.90%      416MB 86.58%                | bytes.(*Buffer).ReadFrom
                                          416MB   100% |   bytes.makeSlice
----------------------------------------------------------+-------------
                                          416MB   100% |   runtime.goexit
         0     0% 99.90%      416MB 86.58%                | git.curoverse.com/arvados.git/sdk/go/keepclient.(*
BlockCache).Get.func1
                                          416MB   100% |   io/ioutil.ReadAll
----------------------------------------------------------+-------------
                                          416MB   100% |   git.curoverse.com/arvados.git/sdk/go/keepclient.
(*BlockCache).Get.func1
         0     0% 99.90%      416MB 86.58%                | io/ioutil.ReadAll
                                          416MB   100% |   io/ioutil.readAll
----------------------------------------------------------+-------------
                                          416MB   100% |   io/ioutil.ReadAll
         0     0% 99.90%      416MB 86.58%                | io/ioutil.readAll
                                          416MB   100% |   bytes.(*Buffer).ReadFrom
```

```
----------------------------------------------------+-------------
                                            64MB   100% |   main.(*ThrottledLogger).flusher
         0     0% 99.90%       64MB 13.32%          | main.(*ArvLogWriter).Write
                                            64MB   100% |   main.(*CollectionFileWriter).Write
----------------------------------------------------+-------------
                                            64MB   100% |   main.(*ArvLogWriter).Write
         0     0% 99.90%       64MB 13.32%          | main.(*CollectionFileWriter).Write
                                            64MB   100% |   main.(*CollectionFileWriter).ReadFrom
----------------------------------------------------+-------------
                                            64MB   100% |   runtime.goexit
         0     0% 99.90%       64MB 13.32%          | main.(*ThrottledLogger).flusher
                                            64MB   100% |   main.(*ArvLogWriter).Write
----------------------------------------------------+-------------
         0     0% 99.90%      480MB 99.90%          | runtime.goexit
                                           416MB 86.67% |   git.curoverse.com/arvados.git/sdk/go/keepclient.
(*BlockCache).Get.func1
                                            64MB 13.33% |   main.(*ThrottledLogger).flusher
----------------------------------------------------+-------------
```

With GOGC=50 it seems to peak around 1.2 GB instead of 1.6 GB:

```
21552 arvbox    20   0 1645864 1.170g   5708 S  56.1 10.1   0:07.32 crunch-run
```

#3 - 10/12/2017 09:03 PM - Peter Amstutz

With GOGC=10 it peaks a bit less:

```
29949 arvbox    20   0 1516652 999.5m   5640 S  71.8  8.4   0:04.23 crunch-run
```

#4 - 10/13/2017 01:56 AM - Peter Amstutz

So my working theory is that ReadAll() involves a number of intermediate allocations which are causing memory fragmentation.

Fixes:

- BlockCache reads into a fixed-capacity buffer, to avoid reallocations (this cuts resident memory usage by 1/2 and makes memory footprint extremely stable)
- Limit the crunch-run BlockCache to 2 blocks instead of 4 (cuts footprint by 1/2 again)
- Clear the crunch-run BlockCache after loading the Docker image (probably cuts post-image-load footprint by 1/2 again).

The memory footprint of crunch-run loading a 3.8 GB Docker image goes from using about 1.5 GB to about 350 MB.

12447-crunch-run-leak @ 6ee6e654bc873db10037c735a63697d295ec40cb

#5 - 10/13/2017 01:57 AM - Peter Amstutz

- Status changed from New to In Progress

#6 - 10/13/2017 01:57 AM - Peter Amstutz

- Assigned To set to Peter Amstutz

#7 - 10/13/2017 01:59 PM - Tom Clegg

Good to know that ioutil.ReadAll() can result in so much fragmentation...

In block_cache.go, new Clear() is not safe to call concurrently with Get(). There are two different locking mechanisms here, mtx and setupOnce. Easiest fix is to lock mtx in setup() instead of Clear() -- then it would be safe for a caller to call Get and Clear concurrently. More complete fix:

- Get rid of setupOnce and setup
- Clear() locks mtx, and does "c.cache = nil"
- Get() (after locking mtx) makes a new map if c.cache == nil

In crunchrun.go, assuming the -memprofile is meant to be kept around:

- Report error on f.Close()
- Wouldn't it be more useful to get a profile *without* doing an artificial runtime.GC()?
- Usage message could mention "...after running the container"

~~It looks like this branch doesn't change the block cache size as described in note-4, or am I missing it? It looks like we'd need to change "if max < defaultMaxBlocks" to "if max < 1" in~~ source:sdk/go/keepclient/block_cache.go#L33 ~~in order for this to work.~~ (edit: this was in a commit that hadn't been pushed yet.)

Can we leave our import blocks in the goimports style (stdlib, then non-stdlib)? You seem to be using some tool/habit that disagrees with goimports (see Emacs and Go) and it's putting a bunch of noise in our git history -- e.g., logging_test.go appears in this diff for the sole purpose of

un-goimportsing the import block.

#8 - 10/13/2017 02:27 PM - Tom Clegg

In block_cache.go, instead of using a bytes.Buffer, a more direct approach would be to call io.ReadFull using the size returned by kc.Get() (which we currently ignore).

```
rdr, size, _, err := kc.Get(locator)
var data []byte
if err == nil {
  data = make([]byte, int(size), BLOCKSIZE)
  _, err2 := io.ReadFull(rdr, data)
  ...
}
```

#9 - 10/16/2017 05:13 PM - Peter Amstutz

Tom Clegg wrote:

> Good to know that ioutil.ReadAll() can result in so much fragmentation...
>
> In block_cache.go, new Clear() is not safe to call concurrently with Get(). There are two different locking mechanisms here, mtx and setupOnce. Easiest fix is to lock mtx in setup() instead of Clear() -- then it would be safe for a caller to call Get and Clear concurrently. More complete fix:
>
> - Get rid of setupOnce and setup
> - Clear() locks mtx, and does "c.cache = nil"
> - Get() (after locking mtx) makes a new map if c.cache == nil

Done.

> In crunchrun.go, assuming the -memprofile is meant to be kept around:
>
> - Report error on f.Close()

Done.

> - Wouldn't it be more useful to get a profile *without* doing an artificial runtime.GC()?

I'm just following the suggested usage from https://golang.org/pkg/runtime/pprof/

> - Usage message could mention "...after running the container"

Done.

> Can we leave our import blocks in the goimports style (stdlib, then non-stdlib)? You seem to be using some tool/habit that disagrees with goimports (see Emacs and Go) and it's putting a bunch of noise in our git history -- e.g., logging_test.go appears in this diff for the sole purpose of un-goimportsing the import block.

Using goimport now. Fixed.

> In block_cache.go, instead of using a bytes.Buffer, a more direct approach would be to call io.ReadFull using the size returned by kc.Get() (which we currently ignore).

Done, although this ended up being a bit more fussy than I expected. It turned up a bug where we didn't require Content-Length in KeepClient.Get(), which would lead to make([]bytes, -1) in BlockCache, which would panic. So KeepClient.Get() now enforces that there is a valid content length and returns an error if not. (And fixed the test stub that returned keep blocks without setting content-length).

#10 - 10/16/2017 05:51 PM - Peter Amstutz

Passing tests now https://ci.curoverse.com/job/developer-run-tests-remainder/488/

#11 - 10/16/2017 06:00 PM - Tom Clegg

LGTM @ d0414ca72, thanks

#12 - 10/16/2017 06:05 PM - Peter Amstutz

- Status changed from In Progress to Resolved

- % Done changed from 0 to 100

Applied in changeset arvados|commit:15b5b59f5902fdc0fe4eb5366ba3b654b117d7df.