Arvados - Bug #13517

keepstore memory usage appears to not be related to MaxBuffers

05/22/2018 06:35 PM - Joshua Randall

Status:	Resolved	**Start date:**	05/31/2018
Priority:	Normal	**Due date:**	
Assigned To:	Tom Clegg	**% Done:**	100%
Category:	Keep	**Estimated time:**	0.00 hour
Target version:	2018-06-06 Sprint		

Description

Our keepstore nodes have repeatedly run our instances out of memory when under heavy load, resulting in the OOM killer being invoked.

We started out with MaxBuffers set to be close to the amount of memory on the machine (i.e. 126 * 64MiB = 8064MiB with 8.4GiB on the machine). We then repeatedly reduced it, with apparently no effect on keepstore's memory consumption.

We now have MaxBuffers: 64 (which should be 4GiB of RAM) on an 8.4GiB machine, and it keeps running the instances out of RAM. Sometimes when the OOM killer decides to kill keepstore, we get a clear record of it being responsible for consuming pretty much all of the RAM on the instance:

```
[19519.435919] Out of memory: Kill process 13297 (keepstore) score 911 or sacrifice child
[19519.437758] Killed process 13297 (keepstore) total-vm:9871532kB, anon-rss:8014404kB, file-rss:1104kB
```

Subtasks:

Task # 13551: Review 13517-buffer-leak	**Resolved**

Associated revisions

Revision 1bff2ab0 - 05/31/2018 07:05 PM - Tom Clegg

Merge branch '13517-buffer-leak'

refs #13517

Arvados-DCO-1.1-Signed-off-by: Tom Clegg <tclegg@veritasgenetics.com>

History

#1 - 05/22/2018 06:37 PM - Joshua Randall

I've confirmed that our keepstore process is running with `GOGC=10` in its environment.

#2 - 05/22/2018 06:41 PM - Joshua Randall

our config file:

```
root@arvados-keep02-eglyx:~# cat /etc/arvados-keep-config.yaml
# arvados-keep-config.yaml for arvados keep nodes on eglyx
BlobSignatureTTL: 1209600s
BlobSigningKeyFile: /etc/arvados-keep-blob-signing.key
Debug: false
EnableDelete: true
Listen: :25107
LogFormat: json
ManagementToken: ""
MaxBuffers: 64
MaxRequests: 0
PIDFile: ""
RequireSignatures: false
SystemAuthTokenFile: /etc/arvados-data-manager.token
TrashCheckInterval: 24h0m0s
TrashLifetime: 168h0m0s
Volumes:
- AccessKeyFile: /etc/arvados-keep-s3.access_key
  Bucket: arvados-keep-eglyx
```

```
  ConnectTimeout: 0s
  Endpoint: https://cog.sanger.ac.uk
  IndexPageSize: 1000
  LocationConstraint: false
  RaceWindow: 24h0m0s
  ReadOnly: false
  ReadTimeout: 0s
  Region: "generic"
  S3Replication: 3
  SecretKeyFile: /etc/arvados-keep-s3.secret_key
  Type: S3
  UnsafeDelete: true
```

#3 - 05/22/2018 06:41 PM - Joshua Randall

version:

```
root@arvados-keep02-eglyx:~# dpkg -l | grep keepstore
ii  keepstore                        1.1.4.20180510200733-1                     amd64        Keep storage daem
on, accessible to clients on the LAN
```

#4 - 05/22/2018 07:10 PM - Ward Vandewege

Joshua Randall wrote:

> Our keepstore nodes have repeatedly run our instances out of memory when under heavy load, resulting in the OOM killer being invoked.
>
> We started out with MaxBuffers set to be close to the amount of memory on the machine (i.e. 126 * 64MiB = 8064MiB with 8.4GiB on the machine). We then repeatedly reduced it, with apparently no effect on keepstore's memory consumption.
>
> We now have MaxBuffers: 64 (which should be 4GiB of RAM) on an 8.4GiB machine, and it keeps running the instances out of RAM. Sometimes when the OOM killer decides to kill keepstore, we get a clear record of it being responsible for consuming pretty much all of the RAM on the instance:

We usually reserve some extra space; the formula we use (documented at https://doc.arvados.org/install/install-keepstore.html) is

The -max-buffers argument limits keepstore's memory usage. It should be set such that max-buffers * 64MiB + 10% fits comfortably in memory. On a host dedicated to running keepstore, divide total memory by 88MiB to suggest a suitable value. For example, if grep MemTotal /proc/meminfo reports MemTotal: 7125440 kB, compute 7125440÷(88×1024)=79 and configure -max-buffers=79.

So, 64 buffers should be fine. Maybe double check that MaxBuffers in the config is really being applied?

```
curl http://localhost:25107/status.json | jq .
```

That'll tell you "BuffersMax" in the "BufferPool" block. That should line up with your MaxBuffers setting.

Another question, are you only seeing this with version 1.1.4.20180510200733 or also with older versions?

#5 - 05/23/2018 03:53 PM - Joshua Randall

We now have it set to MaxBuffers: 64 on a host with 16GiB and it is still consuming all of the host memory.

It looks like the time when this happens corresponds with the upstream S3 gateway becoming overloaded and being slow and/or not responding for some time.

The keepstore logs contain entries such as:

```
2018-05-23_15:22:09.64045 {"level":"info","msg":"reached max buffers (64), waiting","time":"2018-05-23T15:22:0
9.640431345Z"}
2018-05-23_15:22:09.64045 {"RequestID":"req-bm1afke8jngn6u9mv754","level":"info","msg":"request","remoteAddr":
"10.101.0.126:41086","reqBytes":0,"reqForwardedFor":"","reqHost":"arvados-keep02-eglyx.node.zeta-hgiarvados.co
nsul:25107","reqMethod":"GET","reqPath":"d3bf93a6316141799e9a18e1f5040cc2+67108864+Adf2c5bcfb215167e1e98a5b488
2972d747ce23b0@5b17eccb","reqQuery":"","time":"2018-05-23T15:22:09.640274127Z"}
2018-05-23_15:22:09.64049 {"level":"info","msg":"reached max buffers (64), waiting","time":"2018-05-23T15:22:0
9.640472592Z"}
2018-05-23_15:22:09.64061 {"RequestID":"req-ivnjojaukedo2ax6ylon","level":"info","msg":"request","remoteAddr":
"10.101.0.118:25198","reqBytes":67108864,"reqForwardedFor":"","reqHost":"arvados-keep02-eglyx.node.zeta-hgiarv
ados.consul:25107","reqMethod":"PUT","reqPath":"17213de7380297d3282ac1d05f97f6c6","reqQuery":"","time":"2018-0
5-23T15:22:09.640562090Z"}
2018-05-23_15:22:09.64064 {"level":"info","msg":"reached max buffers (64), waiting","time":"2018-05-23T15:22:0
9.640625896Z"}
2018-05-23_15:22:09.84774 {"RequestID":"req-zvaouknwaqvx6teszsl8","level":"info","msg":"response","remoteAddr"
```

:"10.101.0.15:11328","reqBytes":67108864,"reqForwardedFor":"","reqHost":"arvados-keep02-eglyx.node.zeta-hgiarvados.consul:25107","reqMethod":"PUT","reqPath":"baf59c5fdf67198fee12cfe1cf02bac1","reqQuery":"","respBytes":20,"respStatus":"Service Unavailable","respStatusCode":503,"time":"2018-05-23T15:22:09.847622872Z","timeToStatus":15.006072,"timeTotal":15.006079,"timeWriteBody":0.000007}
2018-05-23_15:22:09.84775 {"level":"info","msg":"waited 207.28484ms for a buffer","time":"2018-05-23T15:22:09.847716455Z"}
2018-05-23_15:22:10.45715 {"RequestID":"req-bor8zhwnpexjkgbjw8fp","level":"info","msg":"response","remoteAddr":"10.101.0.126:40970","reqBytes":0,"reqForwardedFor":"","reqHost":"arvados-keep02-eglyx.node.zeta-hgiarvados.consul:25107","reqMethod":"GET","reqPath":"cb1bd846ee8c65c878c6e0a03948f47a+67108864+A797d6c74f9042c151fa8bd8dc62b6b22d643ba81@5b17edea","reqQuery":"","respBytes":20,"respStatus":"Service Unavailable","respStatusCode":503,"time":"2018-05-23T15:22:10.457033762Z","timeToStatus":14.994317,"timeTotal":14.994330,"timeWriteBody":0.000013}
2018-05-23_15:22:10.45719 {"level":"info","msg":"waited 816.669104ms for a buffer","time":"2018-05-23T15:22:10.457143445Z"}
2018-05-23_15:22:10.79763 {"RequestID":"req-a7hobbbquin6shvw9ddg","level":"info","msg":"request","remoteAddr":"10.101.0.122:21950","reqBytes":67108864,"reqForwardedFor":"","reqHost":"arvados-keep02-eglyx.node.zeta-hgiarvados.consul:25107","reqMethod":"PUT","reqPath":"34a840f01c050e70baab6b1ea08f5a6b","reqQuery":"","time":"2018-05-23T15:22:10.797505490Z"}

It looks like what might be happening is that it only counts the buffers on one half of the connection, and assumes that flushing it to the backend storage will work. In this case, the S3 backend is not able to keep up and so buffers waiting to be flushed to S3 pile up during an S3 outage/slowdown until eventually they consume all memory on the machine.

#6 - 05/23/2018 04:22 PM - Joshua Randall

status as requested:

```
root@arvados-keep02-eglyx:~# curl http://localhost:25107/status.json | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   516  100   516    0     0   452k      0 --:--:-- --:--:-- --:--:--  503k
{
  "Volumes": [
    {
      "Label": "s3-bucket:\"arvados-keep-eglyx\"",
      "Status": {
        "MountPoint": "",
        "DeviceNum": 1,
        "BytesFree": 67108864000,
        "BytesUsed": 1
      },
      "InternalStats": {
        "Errors": 0,
        "InBytes": 92260384888,
        "OutBytes": 0,
        "Ops": 3792,
        "GetOps": 1687,
        "PutOps": 549,
        "HeadOps": 1556,
        "DelOps": 0,
        "ListOps": 0
      }
    }
  ],
  "BufferPool": {
    "BytesAllocatedCumulative": 19126026240,
    "BuffersMax": 64,
    "BuffersInUse": 0
  },
  "PullQueue": {
    "InProgress": 0,
    "Queued": 0
  },
  "TrashQueue": {
    "InProgress": 0,
    "Queued": 0
  },
  "RequestsCurrent": 1,
  "RequestsMax": 128,
  "Version": "1.1.4.20180510200733"
}
```

#7 - 05/31/2018 02:54 PM - Tom Clegg

Joshua Randall wrote:

> It looks like what might be happening is that it only counts the buffers on one half of the connection, and assumes that flushing it to the backend storage will work. In this case, the S3 backend is not able to keep up and so buffers waiting to be flushed to S3 pile up during an S3 outage/slowdown until eventually they consume all memory on the machine.

I think this is about right. More detail:

Buffers are taken from a pool -- but we use a sync.Pool, which doesn't conserve entries (it deletes unused entries during GC, and makes new ones). So if a goroutine is still holding on to a buffer after it's returned to the pool, the pool might discard it, and allocate a new one; now we have two buffers, but we're only counting 1 against our limit.

(*S3Volume)Put() copies buffered data to the s3 client's http request body through an io.Pipe. If the s3 client returns without reading the whole pipe, io.Copy() stays blocked on write, and holds on to the buffer forever.

We do unblock the io.Copy() and release the buffer correctly when the caller (keepstore client) hangs up first -- we need similar treatment for the [partially] unread request body case.

source:services/keepstore/s3_volume.go

```
        select {
        case <-ctx.Done():
                theConfig.debugLogf("%s: taking PutReader's input away: %s", v, ctx.Err())

// Our pipe might be stuck in Write(), waiting for

// io.Copy() to read. If so, un-stick it. This means

// PutReader will get corrupt data, but that's OK: the

// size and MD5 won't match, so the write will fail.
                go io.Copy(ioutil.Discard, bufr)

// CloseWithError() will return once pending I/O is done.
                bufw.CloseWithError(ctx.Err())
                theConfig.debugLogf("%s: abandoning PutReader goroutine", v)
                return ctx.Err()
        case <-ready:
                "BUG IS HERE", "need to call io.Copy(ioutil.Discard, bufr)"
                return v.translateError(err)
        }
```

#8 - 05/31/2018 05:33 PM - Tom Clegg

- Status changed from New to In Progress

- Assigned To set to Tom Clegg

- Target version set to 2018-06-06 Sprint

#9 - 05/31/2018 05:45 PM - Tom Clegg

13517-buffer-leak @ 7e9b9d420b58c22c41a713b9bfbfd0f5718abb1a

#10 - 05/31/2018 07:00 PM - Lucas Di Pentima

Is there a way to simulate this error case to write a test? AFAICS, there's a runtime package that may be useful to query for memory usage. Otherwise, LGTM.

#11 - 06/01/2018 07:09 PM - Tom Clegg

- Status changed from In Progress to Feedback

#12 - 06/06/2018 02:47 PM - Tom Clegg

- Status changed from Feedback to Resolved

#13 - 07/23/2018 06:52 PM - Tom Morris

- Release set to 13