While using the aws cli or trying to get a file using curl, I get an error from the radosgw:

```bash
download failed: s3://mpd/0002/0002-dashevc-main-2s-v1.mp4 to ./0002-dashevc-main-2s-v1.mp4 An error occurred (NoSuchKey) when calling the GetObject operation: Unknown
```

Overall, the cluster looks healthy:

```bash
root@node1-1:~# ceph -s
cluster:
  id: 066f558c-6789-4a93-aaf1-5af1ba01a3ad
  health: HEALTH_WARN
  noout flag(s) set

  services:
    mon: 5 daemons, quorum node1-1,node2-1,node3-1,node4-1,node5-1 (age 3d)
    mgr: node2-1(active, since 3d), standbys: node3-1, node1-1, node4-1, node5-1
    osd: 6 osds: 6 up (since 3d), 6 in (since 13d)
      flags noout
    rgw: 6 daemons active (radosgw.node1-1, radosgw.node2-1, radosgw.node3-1, radosgw.node4-1, radosgw.node5-1, radosgw.node6-1)

  task status:

  data:
    pools: 9 pools, 481 pgs
    objects: 338.74k objects, 1.2 TiB
    usage: 3.7 TiB used, 17 TiB / 21 TiB avail
    pgs: 481 active+clean
```

It also seems like the file is available in the bucket index:

```bash
root@node1-1:~# radosgw-admin bi list --bucket=mpd | grep 0002-dashevc-main-2s-v1.mp4
"idx": "0002/0002-dashevc-main-2s-v1.mp4",
"name": "0002/0002-dashevc-main-2s-v1.mp4",
```
```json
{
  "name": "0002/0002-dashevc-main-2s-v1.mp4",
  "size": 31958035,
  "policy": {
    "acl": {
      "acl_user_map": [
        {
          "user": "perfo",
          "acl": 15
        }
      ],
      "acl_group_map": [
        {
          "group": 1,
          "acl": 1
        }
      ],
      "grant_map": [
        {
          "id": "",
          "grant": {
            "type": {
              "type": 2
            },
            "id": "",
            "email": ",
            "permission": {
              "flags": 1
            },
            "name": "",
            "group": 1,
            "url_spec": ""
          }
        },
        {
          "id": "perfo",
          "grant": {
            "type": {
              "type": 0
            },
            "id": "perfo",
            "email": ",
            "permission": {
              "flags": 15
            },
            "name": "perfo user",
            "group": 0,
            "url_spec": ""
          }
        }
      ],
      "owner": {
        "id": "perfo",
        "display_name": "perfo user"
      }
    }
  },
  "etag": "3bec83e90c05823f71d3ddd7e40f33",
  "tag": "64992646-777f-4214-9a7e-6de270ba3d54.350436.82",
  "manifest": {
    "objs": [],
    "obj_size": 31958035,
    "explicit_objs": "false",
    "head_size": 4194304,
    "max_head_size": 4194304,
    "prefix": ".kPQbrpASUP-I4PXgozEZsXh5gQgo44u_"
}
}```
"rules": [ 
  { 
    "key": 0,
    "val": { 
      "start_part_num": 0,
      "start_ofs": 4194304,
      "part_size": 0,
      "stripe_max_size": 4194304,
      "override_prefix": ""
    }
  } 
],
"tail_instance": "",
"tail_placement": { 
  "bucket": { 
    "name": "mpd",
    "marker": "64992646-777f-4214-9a7e-6de270ba3d54.350469.3",
    "bucket_id": "64992646-777f-4214-9a7e-6de270ba3d54.350469.3",
    "tenant": "",
    "explicit_placement": { 
      "data_pool": "",
      "data_extra_pool": "",
      "index_pool": ""
    }
  },
  "placement_rule": "default-placement"
},
"begin_iter": { 
  "part_ofs": 0,
  "stripe_ofs": 0,
  "ofs": 0,
  "stripe_size": 4194304,
  "cur_part_id": 0,
  "cur_stripe": 0,
  "cur_override_prefix": "",
  "location": { 
    "placement_rule": "default-placement",
    "obj": { 
      "bucket": { 
        "name": "mpd",
        "marker": "64992646-777f-4214-9a7e-6de270ba3d54.350469.3",
        "bucket_id": "64992646-777f-4214-9a7e-6de270ba3d54.350469.3",
        "tenant": "",
        "explicit_placement": { 
          "data_pool": "",
          "data_extra_pool": "",
          "index_pool": ""
        }
      },
      "key": { 
        "name": "0002/0002-dashevc-main-2s-v1.mp4",
        "instance": "",
        "ns": ""
      }
    },
    "raw_obj": { 
      "pool": "",
      "cid": "",
      "loc": ""
    },
    "is_raw": false
  }
},
"end_iter": { 
  "part_ofs": 4194304,
  "stripe_ofs": 29360128,
  "ofs": 31958035,
  "placement_rule": "default-placement"
}
Here is the OSD mapping of this object's chunks:

root@node1-1:~# for i in $(seq 0 7); do ceph osd map default.rgw.buckets.data 64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_${i}; done

osdmap e2269 pool 'default.rgw.buckets.data' (11) object '64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_0' -> pg 11.6213653 (11.53) -> up ([5,2,1], p5) acting ([5,2,1], p5)

osdmap e2269 pool 'default.rgw.buckets.data' (11) object '64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_1' -> pg 11.7615522a (11.2a) -> up ([4,3,1], p4) acting ([4,3,1], p4)

osdmap e2269 pool 'default.rgw.buckets.data' (11) object '64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_2' -> pg 11.687dc40 (11.40) -> up ([0,5,1], p0) acting ([0,5,1], p0)

osdmap e2269 pool 'default.rgw.buckets.data' (11) object '64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_3' -> pg 11.ceedb28d (11.8d) -> up ([3,2,5], p3) acting ([3,2,5], p3)

osdmap e2269 pool 'default.rgw.buckets.data' (11) object '64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow__.kPQbrpASUP-I4PXgozEZsXh5gQgo44u_4' -> pg 11.81caf2f4 (11.4f) -> up ([5,1,0], p5) acting ([5,1,0], p5)
However, from the radosgw logs, it seems like some objects are seen as missing on the primary osds:

```
2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 rados->get_obj_iterate_cb oid=64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow_.kPQbrpASUP-I4PXgozEZxH5gQo44u_3 obj-ofs=12582912 read_ofs=0 len=4194304
2020-10-12T10:47:01.585+0000 7f9bd7f57700 1 -- 172.17.5.11:0/1875369112 --> [v2:172.17.5.31:6800/216163,v1:172.17.5.31:6801/216163] -- osd_op(unknown.0.0:449201 11.8d 11:b14db573:::64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow_.kPQbrpASUP-I4PXgozEZxH5gQo44u_3:head [read 0~4194304] snapshot=0=[] ondisk+read+known_if_redirected e2263) v8 -- 0x7f9c68024720 con 0x5603cb2e4290
2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 RGWObjManifest::operator++(): rule->part_size=0 rules.size()=1
2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 RGWObjManifest::operator++(): result: ofs=16777216 start_ofs=16777216 part_ofs=4194304 rule->part_size=0
```

```
However, from the radosgw logs, it seems like some objects are seen as missing on the primary osds:

2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 rados->get_obj_iterate_cb oid=64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow_.kPQbrpASUP-I4PXgozEZxH5gQo44u_3 obj-ofs=12582912 read_ofs=0 len=4194304
2020-10-12T10:47:01.585+0000 7f9bd7f57700 1 -- 172.17.5.11:0/1875369112 --> [v2:172.17.5.31:6800/216163,v1:172.17.5.31:6801/216163] -- osd_op(unknown.0.0:449201 11.8d 11:b14db573:::64992646-777f-4214-9a7e-6de270ba3d54.350469.3__shadow_.kPQbrpASUP-I4PXgozEZxH5gQo44u_3:head [read 0~4194304] snapshot=0=[] ondisk+read+known_if_redirected e2263) v8 -- 0x7f9c68024720 con 0x5603cb2e4290
2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 RGWObjManifest::operator++(): rule->part_size=0 rules.size()=1
2020-10-12T10:47:01.585+0000 7f9bd7f57700 20 RGWObjManifest::operator++(): result: ofs=16777216 start_ofs=16777216 part_ofs=4194304 rule->part_size=0
```

```
```
What surprises me though is that the file is indeed present on the cluster:

```
root@node1-1:~# rados -p default.rgw.buckets.data get 64992646-777f-4214-9a7e-6de270ba3d54.350469.3_0002/0002-dashevc-main-2s-v1.mp4 out
```

The out file is well retrieved and fine so it looks like the file is still there on the cluster.

The issue seems to be with the "shadow" files.

**Related issues:**
- Related to rgw - Feature #49063: rgw: tooling to locate rgw objects with miss... - Pending Backport
- Precedes rgw - Bug #48330: rgw: fix GC defer process and clean up from immedi... - New
- Copied to rgw - Backport #48331: octopus: Object not found on healthy cluster - Resolved

**History**

**#1 - 11/12/2020 01:05 PM - Janek Bevendorff**

Can you check that the RADOS object is larger than 0 bytes? This is a creeping data loss that has been reported on Octopus in several places now and we have experienced it as well. All of a sudden, objects disappear, but their metadata remain.

For us, this issue has manifested itself as follows: Of the 78 buckets we have (having 10s and 100s of thousands of objects each), two are suddenly missing objects. S3 and `radosgw-admin stat` can still find the objects' metadata, but the payload object is gone. Via the S3 interface, this shows as a 404 error (NoSuchKey).

All affected objects are multipart objects, but that may be a coincidence. If I run `rados stat` on the head object, it shows a size of 0 bytes. All subsequent shadow objects are gone completely. I pulled a dump of all objects in my RGW buckets pool (61GB of object names) and ack'ed it for the shadow suffixes shown in `radosgw-admin stat`. I cannot find any of them.

So far, we have lost 16 objects this way. This number hasn't increased in the last two days, but I can only expect this data loss to continue silently in the background. This is a VERY serious issue and I don't know how to debug it.

**#2 - 11/12/2020 02:50 PM - Casey Bodley**

- Project changed from Ceph to rgw
- Priority changed from Normal to Urgent

**#3 - 11/12/2020 03:12 PM - Casey Bodley**

Do you have any other storage classes configured? Is there a lifecycle policy on the bucket?

**#4 - 11/12/2020 03:17 PM - Casey Bodley**

- Status changed from New to Triaged

**#5 - 11/12/2020 03:45 PM - Mark Kogan**

May we also please ask to attach the ceph.conf and `radosgw-admin object stat` of a similarly sized OK object from the same bucket.

**#6 - 11/12/2020 05:04 PM - Janek Bevendorff**

- File radosgw-admin-stat.json.gz added
I am not running Ceph in Kubernetes.

I attached a config dump and a stat dump of an OK object. All sensitive information has been XXXed out.

---

#7 - 11/12/2020 05:11 PM - Janek Bevendorff

I have not set any lifecycle policy on the bucket and I double checked that there is indeed none. `s3cmd getlifecycle s3://bucket` gives `NoSuchLifecycleConfiguration`.

---

#8 - 11/12/2020 09:13 PM - J. Eric Ivancich

I did some experiments, and I don't think the head object having a size of 0 is necessarily an indicator of a problem. That happens whenever a multipart upload is invoked.

If multipart upload is **not** invoked, and the object is either small or requires shadow objects, the head object will contain all the data or the first stripe of the data.

---

#9 - 11/13/2020 08:18 AM - Janek Bevendorff

- File `radosgw-admin-stat.json.gz` added

Indeed. I just noticed the other object isn't a multipart object. I attached the `radosgw-admin` stat dump of a working multipart object. When I `rados stat` the head object, it also reports 0 bytes. But I can find all the shadow objects in my `rados ls` dump, which is not the case for the broken objects.

---

#10 - 11/13/2020 07:16 PM - J. Eric Ivancich

I'm copying in answers Janek gave to questions I posted on ceph-users.

1. It seems like those reporting this issue are seeing it strictly after upgrading to Octopus. From what version did each of these sites upgrade to Octopus? From Nautilus? Mimic? Luminous?

   I upgraded from the latest Luminous release.

2. Does anyone have any lifecycle rules on a bucket experiencing this issue? If so, please describe.

   Nope.

3. Is anyone making copies of the affected objects (to same or to a different bucket) prior to seeing the issue? And if they are making copies, does the destination bucket have lifecycle rules? And if they are making copies, are those copies ever being removed?

   We are not making copies, but we have bucket ACLs in place, which allow different users to access the objects. I doubt this is the problem though, otherwise we probably would have lost terabytes upon terabytes and not 16 objects so far.

4. Is anyone experiencing this issue willing to run their RGWs with `debug_ms=1`? That would allow us to see a request from an RGW to either remove a tail object or decrement its reference counter (and when its counter reaches 0 it will be deleted).

   I haven't had any new data loss in the last few days (at least I think so, I read 1 byte from all objects, but didn't compare checksums, so I cannot say if all objects are complete, but at least all are there).
@Julien Demais and ANYONE ELSE experiencing this issue,

I posted these questions on ceph-users but wanted to see if you would post answers here in the tracker.

1. It seems like those reporting this issue are seeing it strictly after upgrading to Octopus. From what version did each of these sites upgrade to Octopus? From Nautilus? Mimic? Luminous?

2. Does anyone have any lifecycle rules on a bucket experiencing this issue? If so, please describe.

3. Is anyone making copies of the affected objects (to same or to a different bucket) prior to seeing the issue? And if they are making copies, does the destination bucket have lifecycle rules? And if they are making copies, are those copies ever being removed?

4. Is anyone experiencing this issue willing to run their RGWs with 'debug_ms=1'? That would allow us to see a request from an RGW to either remove a tail object or decrement its reference counter (and when its counter reaches 0 it will be deleted).

Thanks,

Eric

---

Please find my answers below:

1. It seems like those reporting this issue are seeing it strictly after upgrading to Octopus. From what version did each of these sites upgrade to Octopus? From Nautilus? Mimic? Luminous?

   I am seeing this issue on a cluster directly created with octopus (1.15.3). No upgrade in my case.

2. Does anyone have any lifecycle rules on a bucket experiencing this issue? If so, please describe.

   I did not specify any chile creating the bucket. I had only one bucket created using the aws cli (using default placement).

3. Is anyone making copies of the affected objects (to same or to a different bucket) prior to seeing the issue? And if they are making copies, does the destination bucket have lifecycle rules? And if they are making copies, are those copies ever being removed?

   No one is using my ceph cluster so far. I am in the process of evaluating ceph so I am the only one using it. I am evaluating the read/write performance of ceph as a streaming platform storage solution so my use case was the following:
   - Create a bucket
   - Add a bunch of file there still using the aws s3 cli with public read ACL
   - Create 1500 copies of those files in the same bucket to generate a fictional big content set
   - I then have a jmeter script to simulate lots of multiple parallel https get request to get the files and measuring when the clients would suffer from empty buffer.
   - During the test, I randomly reboot nodes in order to test ceph resiliency.

   My setup is a 6 nodes cluster. Each node has a 4TB SSD drive (so 1 OSD per node) and a radosgw running. Five of those nodes also are MONs and MGRs. I know this is not a recommended deployment but this is all I have.

   I had lots of test rounds working successfully for days/weeks and all of a sudden, the http requests started to return 404 errors on some files.

4. Is anyone experiencing this issue willing to run their RGWs with 'debug_ms=1'? That would allow us to see a request from an RGW to either remove a tail object or decrement its reference counter (and when its counter reaches 0 it will be deleted).

   Sadly, I had to reset my cluster since I created this bug ticket. I will try to recreate this issue though if it can help to find the root cause somehow.
Julien Demais wrote:

No one is using my ceph cluster so far. I am in the process of evaluating ceph so I am the only one using it. I am evaluating the read/write performance of ceph as a streaming platform storage solution so my use case was the following:
- Create a bucket
- Add a bunch of file there still using the aws s3 cli with public read ACL
- Create 1500 copies of those files in the same bucket to generate a fictional big content set
- I then have a jmeter script to simulate lots of multiple parallel https get request to get the files and measuring when the clients would suffer from empty buffer.
- During the test, I randomly reboot nodes in order to test ceph resiliency.

@Julien and @Janek
Trying to generate a similar workload to reproduce.

A clarification question please to narrow the affected objects.

Does the problem occur only on Shadow objects (NON Multipart) like the originally reported object s3://mpd/0002/0002-dashevc-main-2s-v1.mp4

```
"ns": "shadow"
```

or only on Multipart objects

```
"ns": "multipart"
```

or both object types?

---

Janek Bevendorff wrote:

Indeed. I just noticed the other object isn't a multipart object. I attached the radosgw-admin stat dump of a working multipart object. When I rados stat the head object, it also reports 0 bytes. But I can find all the shadow objects in my rados ls dump, which is not the case for the broken objects.
@Janek, thanks

Noticed that lz4 compression was enabled in the attached json,

```
  "compression": {
    "compression_type": "lz4",
  }
```

few questions please:
- does the issue reproduce on objects that were not compressed (or not compressed with lz4)?
- is it possible that there are multiple RGWs and not all were restarted after the compression was enabled?

#15 - 11/16/2020 08:40 AM - Janek Bevendorff

So far, the issue has only affected multipart objects. I have already suspected that this might be part of the issue, but I cannot be sure. The data loss is steady, but slow, so it's hard to say anything for sure. After four days of no data loss, we lost another 6 objects over the weekend, totalling 19 lost S3 objects now. The pool has a total of 162 million RADOS objects, so the percentage is quite low, but expanding.

We will try to reupload all objects as single-part objects now as a direct counter measure, which with a good portion of luck will stop this from happening.

The LZ4 compression is another thing I already suspected (especially considering the previous and now fixed bug where LZ4 caused OSDs to crash due to unexpected debug output). I have disabled it for now, but cannot make any statements about what would happen to uncompressed objects at this time. We have had LZ4 compression enabled from the start, so all objects are compressed. We do have multiple RadosGWs running (7 to be precise), but they are all in sync, since compression has been on since way before the upgrade from Nautilus. I don't believe we ever had an inconsistent setup here.

What I do remember is error messages about compression during OSD upgrades (i.e., compression on RADOS level) from Nautilus. The errors silently disappeared, though, and the OSDs started normal and everything was fine for a while. I just rechecked the logs and I can see some messages like this:

```
2020-11-13T08:45:40.227+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 2333323974 in db/084544.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.319+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 177483464 in db/084545.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.367+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 177483472 in db/084546.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.415+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 184561352 in db/084547.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.467+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 181186248 in db/084548.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.507+0100 7f4cd3a63700 3 rocksdb: [table/block_based_table_reader.cc:1117] Encountered error while reading data from compression dictionary block Corruption: block checksum mismatch: expected 0, got 177483464 in db/084549.sst offset 18446744073709551615 size 18446744073709551615
2020-11-13T08:45:40.579+0100 7f4cd3a63700 4 rocksdb: [db/compaction_job.cc:1395] [default] [JOB 1363] Compacted 1@2 + 6@3 files to L3 => 274404171 bytes
2020-11-13T08:45:40.583+0100 7f4cd3a63700 4 rocksdb: [Original Log Time 2020/11/13-08:45:40.585011] [db/compaction_job.cc:768] [default] compacted to: files[0 0 27 228 232 0 0] max score 1.08, MB/sec: 7.4rd, 7.3 wr, level 3, files in(1, 6) out(7) MB in(65.1, 200.3) out(261.7), read-write-amplify(8.1) write-amplify(4.0) OK, records in: 7401648, records dropped: 98 output_compression: NoCompression
```

What I do remember is error messages about compression during OSD upgrades (i.e., compression on RADOS level) from Nautilus. The errors silently disappeared, though, and the OSDs started normal and everything was fine for a while. I just rechecked the logs and I can see some messages like this:
This is from only a few days ago, so definitely after the upgrade.

#16 - 11/16/2020 08:56 AM - Dan van der Ster
Has #42670 been excluded?

#17 - 11/16/2020 05:20 PM - Janek Bevendorff
Interesting issue, but it doesn't really look like it's the same thing for two reasons:

1) We never saw this kind of data loss in Nautilus (the linked issue is a Nautilus issue and apparently has a fix already).
2) Objects disappear entirely without user intervention. Once uploaded, our objects are basically immutable and we are only trying to read them, not append anything.

#18 - 11/16/2020 09:33 PM - J. Eric Ivancich
Janek Bevendorff wrote:

What I do remember is error messages about compression during OSD upgrades (i.e., compression on RADOS level) from Nautilus. The errors silently disappeared, though, and the OSDs started normally and everything was fine for a while. I just rechecked the logs and I can see some messages like this:
I tried to trace this to see if it could be related.

There are references to this error message in: https://www.mail-archive.com/ceph-users@ceph.io/msg04003.html

And Igor Fedotov says this is the tracker that root causes and resolves the issue: https://tracker.ceph.com/issues/45613

So you may want to look there and make sure your Octopus is up to date.

#19 - 11/17/2020 02:11 AM - J. Eric Ivancich

Dan van der Ster wrote:

Has #42670 been excluded?

So the first version of Octopus that that fix appears in is 15.2.5, which was released on/about September 16, about 2 months ago.

Can everyone list which is the latest version of Octopus they've experienced this issue in?

Thanks!

#20 - 11/17/2020 08:20 AM - Janek Bevendorff

I waited for the first few minor releases and upgraded from Nautilus directly to 15.2.5, so yes.

#21 - 11/18/2020 04:25 AM - J. Eric Ivancich

In Comment #11 I asked: Is anyone experiencing this issue willing to run their RGWs with 'debug_ms=1'?

Let me follow on with what someone could look for in their logs if they were willing to change this configuration to help us track down the cause of this issue.

RGW can initiate a delete by sending an op to the OSD, and those ops are "refcount.put" and "rgw.obj_remove".

So you'd want to look for log lines that contain either of those two strings PLUS the bucket_id for the bucket. (You can get the bucket id for a bucket with `radosgw-admin metadata get bucket:<bucket-name>`.)

Assuming the bucket_id is 04e65185-554b-4867-8f44-786bd8987ab8.24130.1 and that --debug_ms=1, then I think an egrep along the lines of:

```
egrep '04e65185-554b-4867-8f44-786bd8987ab8.24130.1.*(refcount.put|rgw.obj_remove)' ...
```

06/05/2021
should work. (It might be even more efficient if all the "."s except the one next to *** were changed to ";"; then it would look for a literal "." rather than match any 1 character.)

If you find such lines for a bucket you're not expecting to delete from, I'd like to see a chunk of the logs from at least 1 minute before to at least 1 minute after. And if you find many examples, perhaps one example of rgw.obj_remove and one example of refcount.put to start off with.

I realize there may be some private information in the logs, and if so, please look into ways you might "cleanse" the logs. I'm happy to discuss that topic further.

#22 - 11/18/2020 08:28 AM - Janek Bevendorff
I have enabled the setting via

```bash
ceph config set client.rgw debug_ms 1
```

I will check the logs when the next object vanishes. This doesn't happen too often, though. I only hope the logs won't overflow until then.

#23 - 11/18/2020 08:33 AM - Janek Bevendorff
Short question here: isn't 1 the default log level anyway? I couldn't find any such messages in previous logs at least.

#24 - 11/18/2020 10:20 AM - Dan van der Ster
Janek Bevendorff wrote:

Short question here: isn't 1 the default log level anyway? I couldn't find any such messages in previous logs at least.

The default debug_ms is 0/1, which means it **prints** at level 0 but logs in memory at level 1 (in case of an assertion, the last 10000 memory log entries are dumped to the log file).

If you have debug_ms = 1 (i.e. 1/1) then you'll definitely notice, because the rgw log file will now be spammed with a line for every single message passed with the cluster.

#25 - 11/18/2020 10:21 AM - Dan van der Ster
Janek Bevendorff wrote:

I have enabled the setting via

```bash
[...]
```

I will check the logs when the next object vanishes. This doesn't happen too often, though. I only hope the logs won't overflow until then.
If that doesn't appear to be working, you probably need to do `ceph config set rgw debug_ms 1`

---

### #26 - 11/18/2020 12:54 PM - Janek Bevendorff

The config says the default for rgw is 1/5, that's why I was wondering. [https://docs.ceph.com/en/latest/rados/troubleshooting/log-and-debug/](https://docs.ceph.com/en/latest/rados/troubleshooting/log-and-debug/)

If that doesn't appear to be working, you probably need to do `ceph config set rgw debug_ms 1`.

rgw isn't a valid target. Only client-rgw works. I haven't seen any messages so far, though.

---

### #27 - 11/18/2020 12:59 PM - Dan van der Ster

Janek Bevendorff wrote:

> The config says the default for rgw is 1/5, that's why I was wondering. [https://docs.ceph.com/en/latest/rados/troubleshooting/log-and-debug/](https://docs.ceph.com/en/latest/rados/troubleshooting/log-and-debug/)

> [...]  

> rgw isn't a valid target. Only client-rgw works. I haven't seen any messages so far, though.

hmm. indeed ceph config is inconsistent sometimes.

You can try the old school way to change options at runtime:

`ls /var/run/ceph/*.asok` -- get the asok of the running rgw, then

```bash
ceph --admin-daemon /path/to/asok config set debug_ms 1
```

---

### #28 - 11/18/2020 01:07 PM - Janek Bevendorff

I believe client-rgw is doing the job. The default here seems to be 0/0 and when I turn it back off, I have nothing in my RadosGW logs. With it set to 1, I have a constant stream of osd_op_" messages.

---

### #29 - 11/18/2020 04:19 PM - Denis Krienbühl

We're hitting this problem in our 15.2.5 cluster as well. So far we know of one S3 that went missing, in a typical Docker registry bucket (no lifecycle,
no versioning, multi-part uploads).

We tried setting `debug_ms 1`, but the amount of log that creates is too much for us to handle. We are regularly running `gc list --include-all` to hopefully catch it deleting the next object when that happens.

Apart from that we are trying to go through all our multipart objects expected on the buckets, comparing them to the objects on rados. This will take a long time to complete though.

This is a very serious issue for us, if there's anything we can do to help speed this along, please let us know. This includes financial contributions.

---

**#30 - 11/19/2020 10:26 AM - Denis Krienbühl**

An update on our efforts: We enabled debug_ms 1 and we are trying to find an object that vanishes while these logs are active.

We are also checking our __multipart__ Rados objects. We found that for S3 objects whose HEAD works, but whose GET produce NoSuchKey, all related __multipart__ objects are gone. They show up in radosgw-admin bucket radoslist, but are not found by rados stat.

So far we only have one bucket where this occurs, but that's probably too early to call a data point.

---

**#31 - 11/20/2020 08:12 AM - Denis Krienbühl**

Another update from our side: We have two sites, a larger and a smaller one. The smaller site was successfully scanned yesterday, without any missing objects.

The second site however brought forth a second bucket where this occurred. There are a two correlations that are noteworthy, if again interpreted with caution:

1. Both buckets are from the same customer, but they are driven by different software. One is a Docker registry, one is a basic bucket to serve downloads of large-ish files. Neither have versioning nor lifecycles active. From what we can tell there's nothing special about them.

2. The upload dates of the files that went missing cluster suspiciously around a time when the customer had network issues with intermittent packet loss:

<table>
<thead>
<tr>
<th>Last mod</th>
<th>GMT</th>
<th>Bucket Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon, 31 Aug 2020</td>
<td>21:16:33 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Wed, 21 Oct 2020</td>
<td>14:52:36 GMT</td>
<td>regular bucket</td>
</tr>
<tr>
<td>Thu, 22 Oct 2020</td>
<td>09:12:31 GMT</td>
<td>regular bucket</td>
</tr>
<tr>
<td>Thu, 22 Oct 2020</td>
<td>21:41:18 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Thu, 22 Oct 2020</td>
<td>21:41:38 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Thu, 29 Oct 2020</td>
<td>22:41:54 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Fri, 30 Oct 2020</td>
<td>18:07:38 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Fri, 30 Oct 2020</td>
<td>20:11:46 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:27:02 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:27:03 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:40:41 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:41:02 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:42:49 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>18:43:22 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>20:55:20 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>20:55:24 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>20:55:48 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>20:58:17 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Mon, 02 Nov 2020</td>
<td>23:06:45 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Tue, 03 Nov 2020</td>
<td>00:08:08 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Tue, 03 Nov 2020</td>
<td>00:09:03 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Tue, 03 Nov 2020</td>
<td>00:09:40 GMT</td>
<td>registry bucket</td>
</tr>
<tr>
<td>Sun, 08 Nov 2020</td>
<td>14:11:02 GMT</td>
<td>regular bucket</td>
</tr>
</tbody>
</table>

---

06/05/2021
The dates on November 2nd and 3rd coincide with those network issues.

I don't think that's enough to be sure that packet-loss could be at the heart of this issue, but it does certainly stand out from our data, that we currently only know of a single customer, that tends to access our object storage across the globe where packet-loss might play a role.

This isn't a lot to go on, but maybe it sparks an idea about what could be the cause of this. Note that we are still scanning our bigger site and we might find other examples that contradict this.

We also want to start checking all the actions taken by the garbage collector, as seen in the debug log via rgw.obj_remove, to hopefully find an instance where this occurs as it happens, so we can provide a detailed log.

#32 - 11/20/2020 08:30 AM - Janek Bevendorff
I don't think the issue is packet loss per se. The objects in our affected buckets were all filled by a Hadoop job running on a nearby compute cluster. There may have been interruptions of some sort, but surely they weren't due to packet loss (there may have been timeouts, reschedules, preemptions).
What is weird is that most of the objects are single-part, but a (very) small percentage are in fact multi-part and I can't figure out why. It must be to do with they way Hadoop's s3a driver works. We never actively chose to upload those as multi-part objects.

#33 - 11/20/2020 08:34 AM - Denis Krienbühl
Janek Bevendorff wrote:

I don't think the issue is packet loss per se. The objects in our affected buckets were all filled by a Hadoop job running on a nearby compute cluster. There may have been interruptions of some sort, but surely they weren't due to packet loss (there may have been timeouts, reschedules, preemptions).
What is weird is that most of the objects are single-part, but a (very) small percentage are in fact multi-part and I can't figure out why. It must be to do with they way Hadoop's s3a driver works. We never actively chose to upload those as multi-part objects.

Oh, so the S3 objects that are missing on your side are mainly single-part objects? We've been focusing on mulit-part as the initial files we found missing were multi-part. If you see single-part objects as well we might want to start and search that too.

#34 - 11/20/2020 08:34 AM - Denis Krienbühl
Janek Bevendorff wrote:

I don't think the issue is packet loss per se. The objects in our affected buckets were all filled by a Hadoop job running on a nearby compute cluster. There may have been interruptions of some sort, but surely they weren't due to packet loss (there may have been timeouts, reschedules, preemptions).
What is weird is that most of the objects are single-part, but a (very) small percentage are in fact multi-part and I can't figure out why. It must be
to do with the way Hadoop's s3a driver works. We never actively chose to upload those as multi-part objects.

Oh, so the S3 objects that are missing on your side are mainly single-part objects? We've been focusing on multi-part as the initial files we found missing were multi-part. If you see single-part objects as well we might want to start and search that too.

#35 - 11/20/2020 08:55 AM - Janek Bevendorff
Sorry if that was unclear. The majority of all objects we have in total are single-part objects. The affected objects, however, are all multi-part (19 in total so far).

#36 - 11/20/2020 12:03 PM - Janek Bevendorff
I wanted to reupload our multipart objects as single parts and noticed something very strange: Not all multipart objects identify themselves as such via their ETags. In fact, when I upload a new multipart object via s3cmd, the ETag will ALWAYS be the MD5 sum of the entire object. Many existing multipart ETags, however, look like this: 788a4bc54dd3eece8065d9f96045623d-18, which is, I presume, an iterative hash of the last part combined with the hashes of the previous parts (I cannot use this to check the integrity of the file). I don't know under which circumstances an object receives such an ETag, but I can only assume it does so when a multipart upload interrupted and later on resumed.

That said, ALL our affected objects have an ETag like this. Despite this updated knowledge, what I said earlier still remains true: The buckets contain a mixture of multipart and singlepart objects (yes, also large ones like 1.5GB for some reason, I verified this via radosgw-admin). So far only multipart objects with partial hashes are affected, but now I also have to assume that the actual number of multipart objects may be far larger, since I cannot reliably identify them via S3.

I cannot be too confident here, since my sample size is only 19, but from these observations it appears likely that the issue affects only (or primarily) multipart objects with partial hashes.

#37 - 11/20/2020 12:39 PM - Denis Krienbühl
Janek Bevendorff wrote:

I wanted to reupload our multipart objects as single parts and noticed something very strange: Not all multipart objects identify themselves as such via their ETags. In fact, when I upload a new multipart object via s3cmd, the ETag will ALWAYS be the MD5 sum of the entire object. Many existing multipart ETags, however, look like this: 788a4bc54dd3eece8065d9f96045623d-18 [...] Are you sure that s3cmd actually creates multiple parts in this case? That is, do you see s3cmd doing multiple uploads as follows (here, 5MiB chunks are used).

```
$ s3 s3cmd -c personal.cfg put 10m 's3://denis/test' --multipart-chunk-size-mb 5
upload: '10m' -> 's3://denis/test' [part 1 of 2, 5MB] [1 of 1]
5242880 of 5242880 100% in 1s 4.07 MB/s done
upload: '10m' -> 's3://denis/test' [part 2 of 2, 5MB] [1 of 1]
5242880 of 5242880 100% in 1s 3.13 MB/s done
```

I ask because it would surprise me quite a bit if there are multipart uploads whose ETAG do not contain a dash. From the limited insight I gained while debugging this, it seems that this always holds true. All multipart uploads that I've generated have a dash in their ETag and so do all the objects we lost.
I also found mention of this in different places, I would say that’s a common expectation that this property holds true across multipart uploads.

#38 - 11/20/2020 12:55 PM - Janek Bevendorff

Yes, it is uploading multiple parts and the resulting RADOS objects live in the multipart namespace, not the shadow namespace (verified via radosgw-admin object stat).

I did in fact assume that ALL multipart objects contain dashes, but now I know for sure that’s not really the case. It still surprises me that some or most of our objects are singlepart to begin with, but their proportion is probably a lot smaller than anticipated.

#39 - 11/20/2020 01:18 PM - Denis Krienbühl

Janek Bevendorff wrote:

Yes, it is uploading multiple parts and the resulting RADOS objects live in the multipart namespace, not the shadow namespace (verified via radosgw-admin object stat).

Interesting, what output do you see when you look at the `radosgw-admin bucket radoslist --bucket=<bucket>` output? Assuming you can reproduce this in an empty bucket or one that isn't too large.

We've been looking at the output of this command and there we see something like this:

```
3410862b-8e1e-48be-bf3c-e5e3bef3d757.17968943950.15__multipart_test.2~zfDbcllX30Yms5RQ0-L01G4Xg8U1zM--.1
3410862b-8e1e-48be-bf3c-e5e3bef3d757.17968943950.15__multipart_test.2~zfDbcllX30Yms5RQ0-L01G4Xg8U1zM--.2
3410862b-8e1e-48be-bf3c-e5e3bef3d757.17968943950.15__shadow_test.2~zfDbcllX30Yms5RQ0-L01G4Xg8U1zM--.1_1
3410862b-8e1e-48be-bf3c-e5e3bef3d757.17968943950.15__shadow_test.2~zfDbcllX30Yms5RQ0-L01G4Xg8U1zM--.2_1
3410862b-8e1e-48be-bf3c-e5e3bef3d757.17968943950.15_test
```

For the missing S3 objects the __multipart files cannot be found using rados -p default.rgw.buckets.data stat <object_id>. If you can show me how your radoslist output for the multipart with the unexpected ETag looks like, I might see something that invalidates my assumptions.

As an aside, we've been looking at catching the GC removing such a multipart object by looking for (refcount.put|rgw.obj_remove), but the only __multipart objects we can see being deleted are ones with a .meta suffix, which happens during upload. So I assume that is just a throwaway. The actual __multipart objects seem to also remain long after the S3 object has been deleted, my test multipart objects have not been GC'd it seems.

That being said, if the theory holds true that this is a GC issue, could we somehow catch the GC throwing away a __multipart object when it should not? I thought I might be able to do this by looking at rgw.obj_remove logs, but if I don't see those objects in there that seems to be unfeasible.

We are gathering lots of logs at the moment, but we need a way to catch this error as it happens. If there are any other ideas on how to do this, let me know :)

06/05/2021
Update: It seems like refcount.put is the call that logs the __multipart file about to be deleted.

I created a bucket and uploaded two 1GB multipart objects. Here's what the radoslist looks like:

```
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.1
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.10
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.11
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.12
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.13
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.14
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.15
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.16
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.17
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.18
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__multipart_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.19
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.10_1
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.10_2
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.10_3
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.11_1
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.11_2
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.11_3
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.12_1
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.12_2
6112a2c2-2570-4c6a-8e57-709633df12eb.343478859.2__shadow_WIDE-20110903143858-01167.warc.gz.2~t5YVPyPcWghoE0RlkHoHxz-lNtk4v8b1p.12_3
...```

Same as yours. s3cmd info is this:

```
s3://reupload-test/WIDE-20110924065228-02544.warc.gz (object):
  File size: 10000390
  Last mod: Fri, 20 Nov 2020 11:24:06 GMT
  MIME type: application/gzip
  Storage: STANDARD
  MD5 sum: 4a2244642bf656ae5c29d7859a81d4a4
  SSE: none
  Policy: none
  CORS: none
  ACL: webis: FULL_CONTROL
```
#42 - 11/20/2020 03:03 PM - Denis Krienbühl
Thank you, I'm glad to see the __multipart files in there, as that's what I'm counting on.

And just in case someone finds this useful or interesting, this is the script with which we're scanning for missing files. It is run on a mon which has access to radosgw-admin.

[https://gist.github.com/href/89c717c7bbaf9f87fd64e4346a6cc889](https://gist.github.com/href/89c717c7bbaf9f87fd64e4346a6cc889)

#43 - 11/20/2020 03:04 PM - Janek Bevendorff
Update: forget what I said. When I retrieve the ETag via boto3, I get 57ba90377073200f91b5dbee7aeece6-64 instead. I am pretty sure, s3cmd's "MD5 sum" field used to be the ETag. I am confused.

#44 - 11/20/2020 07:14 PM - J. Eric Ivancich
If the multipart components are themselves larger than the striping size, they two can have shadow tail objects, as shown in Comment #39.

And when you upload via s3cmd and add --disable-multipart, if the object is large enough, it will have shadow tail objects as well. Is that what people above are referring to as "single part'?

I believe we're aware of a case of missing shadow/tail objects when multipart is not involved.

Eric

#45 - 11/20/2020 07:21 PM - J. Eric Ivancich
We realize not everyone is able to do this, but for those who can, here is one approach we're using to try to nail this bug down.

If you have a bucket where things get uploaded and maybe read but rarely deleted, and if you're able to run all RGWs with --debug_ms=1, then we're looking for log entry lines that contain the bucket id and all ops associated with a) removing objects directly, b) decrementing refcount, which possibly results in deletion of a tail object, c) adding an object to GC, d) deferring an object from removal in GC if it's being read.

egrep '<bucket_id>.*(refcount\ PUT|rgw\ obj_remove|gc_set_entry|gc_defer_entry)' ...

The challenge is various aspects of the story may come through different RGWs. So for example, something may be added for gc via one RGW and the refcount decrements through another.

Thanks to everyone who's lending a hand.

Eric

#46 - 11/20/2020 07:30 PM - J. Eric Ivancich
Just to clarify on why this is helpful: "If you have a bucket where things get uploaded and maybe read but rarely deleted"...
We're trying to get away from user-initiated deletes.

On the other hand, for those that are experiencing losses that they can readily identify, then having logs with --debug_ms=1 surrounding the loss would help us independently.

---

**#47 - 11/20/2020 07:43 PM - Denis Krienbühl**

Eric Ivancich wrote:

If you have a bucket where things get uploaded and maybe read but rarely deleted, and if you're able to run all RGWs with --debug_ms=1, then we're looking for log entry lines that contain the bucket id and all ops associated with a) removing objects directly, b) decrementing refcount, which possibly results in deletion of a tail object, c) adding an object to GC, d) deferring an object from removal in GC if it's being read.

\[
\text{egrep} \ \text{"<bucket_id>\.*\(refcount\:put\|rgw\:obj_remove\|gc\_set\_entry\|gc\_defer\_entry\)" ...}
\]

We are currently running all our RGWs of the affected zone with `debug_ms`, producing north of 4k lines per second. Now we are trying to find an object that is lost after the enabling debug logs - with no luck yet.

Since we haven't discovered any such objects, I'm not sure how useful these logs would be for you at the moment, but if you simply want the debug logs of a run-of-the-mill bucket I could create one for you, use it in any pattern you'd like and send you the associated logs afterwards.

---

**#48 - 11/20/2020 08:55 PM - J. Eric Ivancich**

Denis Krienbühl wrote:

Eric Ivancich wrote:

If you have a bucket where things get uploaded and maybe read but rarely deleted, and if you're able to run all RGWs with --debug_ms=1, then we're looking for log entry lines that contain the bucket id and all ops associated with a) removing objects directly, b) decrementing refcount, which possibly results in deletion of a tail object, c) adding an object to GC, d) deferring an object from removal in GC if it's being read.

\[
\text{egrep} \ \text{"<bucket_id>\.*\(refcount\:put\|rgw\:obj_remove\|gc\_set\_entry\|gc\_defer\_entry\)" ...}
\]

We are currently running all our RGWs of the affected zone with `debug_ms`, producing north of 4k lines per second. Now we are trying to find an object that is lost after the enabling debug logs - with no luck yet.

Since we haven't discovered any such objects, I'm not sure how useful these logs would be for you at the moment, but if you simply want the debug logs of a run-of-the-mill bucket I could create one for you, use it in any pattern you'd like and send you the associated logs afterwards.

Yeah, doubtful it's helpful to examine logs for a run-of-the-mill bucket. I think the two conditions we're looking for are:

a. if you have a bucket that doesn't get deletes then using the egrep command above will help us find the interesting events quickly.

b. if you know of a case of an object that's damaged and we have logs surrounding that event.

We'll try to see if we can find more angles to hit this from.
Thank you for the help!

Eric Ivancich wrote:

a. if you have a bucket that doesn't get deletes then using the egrep command above will help us find the interesting events quickly.

b. if you know of a case of an object that's damaged and we have logs surrounding that event.

Just so I don't get this wrong, you want both of those conditions to be true, right?

That is probably not too likely to happen, but that being said the second bucket where this did occur is one such bucket, and we are keeping those buckets under close observation, so maybe we get lucky and we can capture this in action.

Let me know if we can help you with anything else. This has our full attention.

Eric Ivancich wrote:

egrep "<bucket_id>.*(refcount|put|rgw.obj_remove|gc_set_entry|gc_defer_entry|rgw.gc_queue_enqueue|rgw.gc_queue_update_entry)" ...

Thanks,
Eric

Denis Krienbühl wrote:

Eric Ivancich wrote:

a. if you have a bucket that doesn't get deletes then using the egrep command above will help us find the interesting events quickly.

b. if you know of a case of an object that's damaged and we have logs surrounding that event.

Just so I don't get this wrong, you want both of those conditions to be true, right?

That is probably not too likely to happen, but that being said the second bucket where this did occur is one such bucket, and we are keeping those buckets under close observation, so maybe we get lucky and we can capture this in action.

Let me know if we can help you with anything else. This has our full attention.

The a) case is dependent on a bucket in which removes don't happen (or are incredibly rare), otherwise there will be too many false positives. But if
you have such a bucket, it might find these cases early.

The b) case is the more general one that doesn't need a no-delete bucket. But if you find the case of a damaged object then we might have days worth of logs to see the events that led to it.

So a) and b) are independent but a) is more highly constrained and not everyone will have such a bucket handy.

Eric

#52 - 11/20/2020 09:34 PM - Denis Krienbühl
I see, I'll try to run a search for case "a" next week. We have plenty of access logs going back a few months detailing all HTTP requests against all of our buckets. I might be able to find a bucket that does not get any or few DELETE requests.

#53 - 11/21/2020 04:09 PM - J. Eric Ivancich
Based on a review of code paths, I have a theory as to why this happens. There's an immediate configuration change that, if this theory is correct, would mitigate the data loss.

RGW has a garbage collection (GC) deferral process. It is meant to handle cases of:

1. Read of large object begins
2. Delete issued for large object
3. Read of large object continues

When reads continue beyond 1/2 of the time set in the configuration option rgw_gc_obj_min_wait, a GC defer operation is sent to prevent that object from being GC’d while reads continue.

The GC code was substantially updated for Octopus to use a more efficient back end. The theory is that it introduced a bug such that when GC is deferred, it can add a GC entry for an object that was never deleted. And that deletion will then occur at a later time as the GC queue is processed.

So the quick work-around is to increase rgw_gc_obj_min_wait from a default of 2 hours to a higher value? 5 hours? 10 hours? The choice is yours.

If you set it to 5 hours, then a read would have to exceed 2.5 hours (half the configured time) to trigger deferment and the theorized bug.

Although this is currently theorized and no reproducer has yet been created, this seems to fit the various descriptions of the theorized bug.

1. Tail objects are deleted but head objects are not. That's the behavior of GC.
2. No user-initiated removal of the object occurs; we have cases where a bucket in which deletes are exceedingly rare experience the data loss.
3. The bug occurs rarely; again it's triggered by particularly slow reads. With default config options, the read would have to exceed 1 hour.
4. Someone above thought there was a correlation with network issues; network issues could have slowed owned reads, thus triggering the bug.
5. The bug seems to have started when users upgraded to Octopus, which corresponds to this code change.

We're currently working on a reproducer to verify this theorized bug along with bug-fixes.

If there is any additional evidence, corroborating or invalidating, please post.

Thanks,

Eric
Your theory sounds plausible to me. Though if I understand correctly we might be able to verify this by looking at our requests log, right? I would expect to find a GET request that lasted longer than 1 hour for the objects in question.

If you think that would corroborate your theory let me know, I should be able to get that information.

In any case, I'm still discussing this with my team, but I assume we'll try your suggested workaround.

Denis Krienbühl wrote:

Your theory sounds plausible to me. Though if I understand correctly we might be able to verify this by looking at our requests log, right? I would expect to find a GET request that lasted longer than 1 hour for the objects in question.

If you think that would corroborate your theory let me know, I should be able to get that information.

Yes, that could confirm.

Thank you,

Eric

IMPORTANT

rgw_gc_obj_min_wait is set in seconds.
So 2 hours, the default, is 7200 seconds.
5 hours is 18000 seconds.
And 10 hours is 36000 seconds.

06/05/2021

#55 - 11/21/2020 04:28 PM - J. Eric Ivancich
Denis Krienbühl wrote:

Your theory sounds plausible to me. Though if I understand correctly we might be able to verify this by looking at our requests log, right? I would expect to find a GET request that lasted longer than 1 hour for the objects in question.

If you think that would corroborate your theory let me know, I should be able to get that information.

Yes, that could confirm.

Thank you,

Eric
Please note that there is now a PR attached to this Tracker. I still have not been able to reproduce, as using my command-line tools I've not yet been able to slow down a read object operation to create the conditions that would seem to produce this data loss.

This PR is not a full fix. If the theory is correct it should prevent the data loss, which is the most pressing issue. But it might create orphans in similar situations, tail objects that should be removed ultimately but that are not. More specifically, objects that the user has requested be removed and that are on the GC queue and that are then read slowly, the GC entry might be lost, so the objects are not ultimately deleted. And

We do have the `rgw-orphan-list` tool that you'll be able to use to identify them.

And discussions are underway at providing a more complete solution.

I continue to try to reproduce and will report what I find.

Eric

#59 - 11/21/2020 07:58 PM - Denis Krienbühl

Your theory aligns perfectly with our request logs. Here are the longest measured GET requests for all of the S3 objects that went missing:

1:00:53.887000
1:01:12.880000
1:04:42.373000
1:05:27.778000
1:11:46.220000
1:24:54.102000
2:08:10.981000
2:10:46.777000
2:14:52.990000
2:23:11.656000
2:29:08.968000
2:29:35.797000
2:36:49.871000
2:37:54.386000
2:52:56.228000
2:59:15.081000
3:25:31.692000
3:25:32.937000
3:35:50.415000
4:17:14.685000
4:21:26.926000
4:27:04.474000
4:47:05.590000
5:19:33.749000
6:51:13.892000
7:31:06.271000

As you can see, the shortest request is just a bit longer than an hour, and the longest took a whopping 7½ hours.

As a result we will probably increase `rgw_gc_obj_min_wait` to a even higher number, as we recently upgraded our storage and have plenty to spare.
Denis Krienbühl wrote:

Your theory aligns perfectly with our request logs. Here are the longest measured GET requests for all of the S3 objects that went missing:

[...]

As you can see, the shortest request is just a bit longer than an hour, and the longest took a whopping 7½ hours.

As a result we will probably increase rgw_gc_obj_min_wait to a even higher number, as we recently upgraded our storage and have plenty to spare.

Wow! That's very helpful. Thank you.

Do you have the means to rebuild your RGWs with the PR attached to this tracker? If so that would be an option. Depending on the number of deletes your system sees the larger rgw_gc_obj_min_wait value might cause the GC log to hit the end of its allotted space in which case, GC won't be used and objects will be deleted straight away at a performance cost (noticeable but not significant, I'm told).

Eric

Denis Krienbühl wrote:

Do you have the means to rebuild your RGWs with the PR attached to this tracker? If so that would be an option. Depending on the number of deletes your system sees the larger rgw_gc_obj_min_wait value might cause the GC log to hit the end of its allotted space in which case, GC won't be used and objects will be deleted straight away at a performance cost (noticeable but not significant, I'm told).

If the GC won't be used, would there be any other problems rather than just performance? As in, is our cluster still safe when that happens? Because we can live with a performance hit for the weekend and look into compiling our own build next week.

In any case, thank you for your continued help. We are grateful to have a likely root cause and workaround at hand!
I made some tests in our lab and can confirm the theory. I was able to reproduce this bug. I used the following settings to speed things up:

```shell
debug rgw = 20
ddebug ms = 1
rgw gc obj min wait = 300
rgw gc processor period = 150
```

I uploaded a multipart object and another object which was large enough to have tail "shadow" object without multiparts and slowly downloaded them using curl and restricting the download rate to 10KB/s with "trickle" ([https://github.com/mariusae/trickle](https://github.com/mariusae/trickle)). After a while the shadow and multipart objects showed up in the GC list and were purged by the next GC run after they "expired". With the settings above it took around 10 minutes for them to get deleted.

I also verified that objects without tail objects are not affected by this bug.

If you are still interested in logs, I can probably provide them or look through them for any further hints on what exactly happens. But I would have to clean them a bit before I can attach them to the bug report. Please just tell me if you still need them.

P.S.: I'm a co-worker of Denis.

---

#63 - 11/21/2020 09:40 PM - J. Eric Ivancich

Gaudenz Steinlin wrote:

I made some tests in our lab and can confirm the theory. I was able to reproduce this bug. I used the following settings to speed things up: [...]

I uploaded a multipart object and another object which was large enough to have tail "shadow" object without multiparts and slowly downloaded them using curl and restricting the download rate to 10KB/s with "trickle" ([https://github.com/mariusae/trickle](https://github.com/mariusae/trickle)). After a while the shadow and multipart objects showed up in the GC list and were purged by the next GC run after they "expired". With the settings above it took around 10 minutes for them to get deleted.

I also verified that objects without tail objects are not affected by this bug.

If you are still interested in logs, I can probably provide them or look through them for any further hints on what exactly happens. But I would have to clean them a bit before I can attach them to the bug report. Please just tell me if you still need them.

P.S.: I'm a co-worker of Denis.

Thank you, Gaudenz. That's further confirmation we have the root cause.

I don't think additional logs would be helpful at this point. But I do appreciate the insights you and Denis have brought to the table and your incredible willingness to lend a hand.

Eric
Denis Krienbühl wrote:

Eric Ivancich wrote:

Do you have the means to rebuild your RGWs with the PR attached to this tracker? If so that would be an option. Depending on the number of deletes your system sees the larger rgw_gc_obj_min_wait value might cause the GC log to hit the end of its allotted space in which case, GC won’t be used and objects will be deleted straight away at a performance cost (noticeable but not significant, I’m told).

If the GC won’t be used, would there be any other problems rather than just performance? As in, is our cluster still safe when that happens? Because we can live with a performance hit for the weekend and look into compiling our own build next week.

I am not aware of any reason your cluster would be unsafe. Immediate deletion of tail objects would also make the corner case addressed by GC deferral not possible (1. start read, 2. remove object, 3. continue read). But generally speaking, most applications and users do not expect continued access to an object that’s been deleted.

In any case, thank you for your continued help. We are grateful to have a likely root cause and workaround at hand!

You're very welcome and I appreciate all the help!

Eric

Julien Demais wrote:

This actually makes a lot of sense. Here are what we have set in our config:

- `rgw_gc_obj_min_wait`: 300
- `rgw_gc_processor_max_time`: 600
- `rgw_gc_processor_period`: 600


This was because I tested content load performances and noticed that when deleting the bucket (without bypassing the gc with radosgw-admin), the objects were still there for a very very long time and I ended up filling up my cluster.

In our case, some of the nodes have network issue on boot up so this is very likely that some reads took more than 150 seconds...

Nice work narrowing this down!
This leads my next question: is there a way to get information on how to fine tune the RGW garbage collector? Those settings are pretty "blurry" to me.

#66 - 11/22/2020 03:37 PM - J. Eric Ivancich
- Status changed from Fix Under Review to Pending Backport

#67 - 11/23/2020 04:09 AM - Pritha Srivastava
If the affected users have increased the value of rgw_gc_obj_min_wait, then before restoring the original expiration time once the fixed build is available, it is recommended to clear off the gc queue manually by running - radosgw-admin gc process --include-all. This may be needed since the older entries may have expiration time > newer entries (once the original expiration time is restored), and may delay removal of expired entries leading to a full queue. In order to avoid this, manual gc process can be run to flush out the old entries. Please note that this will remove the gc entries before their expiration time.

#68 - 11/23/2020 01:29 PM - Denis Krienbühl
Pritha Srivastava wrote:

Please note that this will remove the gc entries before their expiration time.

What does this mean for us? We did increase rgw_gc_obj_min_wait and we will have to get back to the default once we update: Would that trigger the bug and therefore cause more data-loss, if we are not careful? That is, do we have to look out for requests that take longer than 1 hour, making sure that we don't have any of those before reverting?

#69 - 11/23/2020 02:47 PM - Pritha Srivastava
Denis Krienbühl wrote:
Pritha Srivastava wrote:

Please note that this will remove the gc entries before their expiration time.

What does this mean for us? We did increase rgw_gc_obj_min_wait and we will have to get back to the default once we update: Would that trigger the bug and therefore cause more data-loss, if we are not careful? That is, do we have to look out for requests that take longer than 1 hour, making sure that we don't have any of those before reverting?

This will not cause any more data loss. What it means is that if objects in gc are to expire and hence get deleted after x seconds from now, if a radosgw-admin gc process --include-all is run, all gc entries will be removed from the system now (which might be before their expiration time) With the fix in place, requests that take longer than one hour won't be added to gc at all.
Pritha Srivastava wrote:

Denis Krienbühl wrote:

Pritha Srivastava wrote:

Please note that this will remove the gc entries before their expiration time.

What does this mean for us? We did increase `rgw_gc_obj_min_wait` and we will have to get back to the default once we update: Would that trigger the bug and therefore cause more data-loss, if we are not careful? That is, do we have to look out for requests that take longer than 1 hour, making sure that we don't have any of those before reverting?

This will not cause any more data loss. What it means is that if objects in gc are to expire and hence get deleted after x seconds from now, if a `radosgw-admin gc process --include-all` is run, all gc entries will be removed from the system now (which might be before their expiration time). With the fix in place, requests that take longer than one hour won't be added to gc at all.

Ah so the best way of doing the update would be to roll out the new version, then change the configuration, right?

Pritha Srivastava wrote:

Denis Krienbühl wrote:

Pritha Srivastava wrote:

Please note that this will remove the gc entries before their expiration time.

What does this mean for us? We did increase `rgw_gc_obj_min_wait` and we will have to get back to the default once we update: Would that trigger the bug and therefore cause more data-loss, if we are not careful? That is, do we have to look out for requests that take longer than 1 hour, making sure that we don't have any of those before reverting?
This will not cause any more data loss. What it means is that if objects in gc are to expire and hence get deleted after x seconds from now, if a radosgw-admin gc process --include-all is run, all gc entries will be removed from the system now (which might be before their expiration time) With the fix in place, requests that take longer than one hour won't be added to gc at all.

Ah so the best way of doing the update would be to roll out the new version, then change the configuration, right?

Yes.

---

#72 - 11/23/2020 06:03 PM - J. Eric Ivancich
- Precedes Bug #48330: rgw: fix GC defer process and clean up from immediate fix added

#73 - 11/23/2020 06:22 PM - J. Eric Ivancich
I exchanged email with Pritha, who is the true expert on modern GC in RGW, to clarify my own questions. I'm pasting that Q&A here for everyone's benefit.

==== Paste Below ====

Hi Pritha,

Just so I'm clear, assume the user:

1. makes rgw_gc_obj_min_wait big (e.g., 10 hours)
2. runs for a while
3. applies the fix
4. manually runs `radosgw-admin gc process --include-all` ...
5. returns rgw_gc_obj_min_wait to the original value (2 hours)
6. objects are deleted as part of the normal running

Now...

If we skip step 4 in the above:

a. the objects deleted in step 6 in the first 10 hours after step 5 might take 10 hours to be gc'd rather than the expected 2 hours.

Yes

b. if the GC queue is full, objects would be deleted in-line rather than GC'd, which might come with a performance hit.
   i. and that may have already happened during step 2 anyway

Yes

c. but otherwise, there would be no problems.

Yes, there would be no problems otherwise.
Is that understanding correct?

By doing the manual GC in step 4 we simply allow the objects deleted in step 6 to only remain on the GC queue for 2 hours. And it would be necessary to use the "--include-all" flag when running manual GC, right?

Yes. It is basically a means to start afresh after restoring the original expiration time because the elements of the queue should have increasing expiration times. Hence I have mentioned this step as a recommendation.

Eric

Thanks,
Pritha
Eric

#79 - 02/04/2021 11:58 PM - J. Eric Ivancich

- Related to Feature #49063: rgw: tooling to locate rgw objects with missing rados components added

Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>radosgw-admin-stat.json.gz</td>
<td>95.2 KB</td>
<td>11/12/2020</td>
<td>Janek Bevendorff</td>
</tr>
<tr>
<td>ceph.conf</td>
<td>14 KB</td>
<td>11/12/2020</td>
<td>Janek Bevendorff</td>
</tr>
<tr>
<td>radosgw-admin-stat.json.gz</td>
<td>18.6 KB</td>
<td>11/13/2020</td>
<td>Janek Bevendorff</td>
</tr>
</tbody>
</table>