Ceph - Feature #4929

Erasure encoded placement group

05/07/2013 04:50 PM - Loic Dachary

Status:	Resolved	**Start date:**	05/16/2013
Priority:	Urgent	**Due date:**	09/20/2013
Assignee:	Loic Dachary	**% Done:**	16%
Category:	OSD	**Estimated time:**	0.00 hour
Target version:		**Spent time:**	146.50 hours
Source:	Development	**Reviewed:**	
Tags:		**Affected Versions:**	
Backport:		**Pull request ID:**	

Description

This is the *home page* of erasure coding implementation in Ceph. The description should contain links to all the background information used by the current developers and required for a new developer to jump in. The sub tasks are a complete inventory of the working currently going on and each of them contains links to the corresponding github pull requests or work in progress.

Documentation:

- Erasure code internal documentation
- FAST Erasure Code tutorial

Firefly Ceph summit:

- video

Emperor Ceph summit:

- http://wiki.ceph.com/01Planning/02Blueprints/Emperor/Erasure_coded_storage_backend_%28step_2%29
- Introduction of the second day, Sage explains the Big Rados Features, including erasure code
  Sage : *... The big one is erasure coding which is great to have more efficient storage and for tiering. It's good for performances and overall for the cost / performance ratio. ... Erasure coding : the motivation here is that erasure coding is a different way to handle data redundancy using much less overall space but you actually get higher overall data durability so you have a lower probabily of losing data when machine dies, disk dies and so forth. Work has already started in this area by Loic and that's been great so over this next two cycles we're going to be putting Inktank resources and our efforts as well : there is a lot of code refactoring, moving around, making sure that it fits nicely into the structure of rados, so we're very excited to get started on that work. ...*

Dumpling Ceph summit documents:

- http://wiki.ceph.com/01Planning/02Blueprints/Dumpling/Erasure_encoding_as_a_storage_backend
- http://pad.ceph.com/p/Erasure_encoding_as_a_storage_backend

Discussions:

- PGBackend proposal
- Erasure code library summary
- Comments on Ceph distributed parity implementation
- Erasure coding implementation : high level description
- Erasure coding library API
- CEPH Erasure Encoding + OSD Scalability
- these conversations ceased to be listed at some point and more can be found when searching in the mailing lists.

Alpha testers:

First testing stage is by running teuthology tests independently, as soon as they include Erasure Code tests.

- Université de Nantes - Yann
- CERN - Andreas
- Rackspace - Darren

- Telekom - Michael
- Cloudapt - Mike
- HTS - Samuel
- five3genomics - Charles

Subtasks:

Subtask # 5046: Factor out PG logs, PG missing	**Resolved**
Subtask # 5085: PG::merge_log should not have side effects other than on the log & miss...	**Rejected**
Subtask # 5213: unit tests for src/osd/PGLog.{cc,h}	**Resolved**
Subtask # 5487: Factor out ObjectContext / ReplicatedPG::object_contexts	**Closed**
Subtask # 5510: ObjectContext : replace ref with shared_ptr	**Resolved**
Subtask # 5527: unit tests for common/sharedptr_registry.hpp	**Resolved**
Subtask # 6119: replace PG::object_contexts with SharedPtrRegistry	**Won't Fix**
Subtask # 5433: Factor out the ReplicatedPG object replication and client IO logic as a...	**Rejected**
Subtask # 5855: Backfill peers should not be included in the acting set	**Resolved**
Subtask # 5856: Refactor Backfill to use PGBackend methods	**Resolved**
Subtask # 5857: Refactor recovery to use PGBackend methods	**Resolved**
Subtask # 5858: Backfill should be able to handle multiple backfill peers	**Resolved**
Subtask # 5859: GetInfo should use PGBackend methods to determine when peering can cont...	**Rejected**
Subtask # 5860: PG::calc_acting and friends should use PGBackend to select the acting s...	**Rejected**
Subtask # 5861: Refactor scrub to use PGBackend methods	**Resolved**
Subtask # 5862: FileStore must work with ghobjects rather than hobjects	**Resolved**
Subtask # 5863: OSD internals must work in terms of cpg_t	**Rejected**
Subtask # 5877: Plugable erasure code library	**Resolved**
Subtask # 5878: erasure plugin mechanism and abstract API	**Resolved**
Subtask # 5879: jerasure plugin	**Resolved**
Subtask # 6113: add ceph osd pool create [name] [key=value]	**Resolved**
Subtask # 6027: ensure that erasure coded pools don't work until the osds can handle it	**Resolved**
Subtask # 6434: review and test PGBackend	**Rejected**
Subtask # 6900: crush: fix indep mode	**Resolved**
Subtask # 6895: omap needs to be disablable on a per-pool basis, this needs to be requi...	**Rejected**
Subtask # 7048: mon: erasure crush rule vs pool process	**Resolved**
Subtask # 6888: EC/Tiering: Disallow omap writes on pools which are backed by EC pools	**Resolved**
Subtask # 7158: EC: flesh out how the ceph tool should be used to manage ec pools and c...	**Resolved**
Subtask # 7292: erasure code: plugin backward compatibility	**Rejected**
Subtask # 7307: erasure-code: chunk_size must not be architecture dependant	**Rejected**
Subtask # 7277: EC: on erasure pool creation, size needs to be fixed to K+M, size must ...	**Resolved**
Subtask # 7146: implement osd crush rule create-erasure	**Resolved**
Subtask # 7313: erasure-code: rule create-erasure requires CEPH_FEATURE_CRUSH_V2	**Resolved**
Subtask # 7339: erasure code: add stripe width to pg_pool_t	**Resolved**

Associated revisions

Revision 980a0738 - 08/09/2013 07:42 AM - Loic Dachary

document the write / read path for erasure encoding

Explains how objects are stored and used in erasure coded pools. It is
the result of discussions that occured on the ceph-devel mailing list
around june 2013. The rationale behind each change can be found in the
archive of the mailing list. For instance, the coding of the chunk
number with the object or the decision to decode using any K chunks
instead of trying to fetch the data chunks when possible because it
would allow simple concatenation when systematic codes are used.

http://tracker.ceph.com/issues/4929 refs #4929

Signed-off-by: Loic Dachary <loic@dachary.org>

Revision 3a831292 - 08/20/2013 03:34 PM - Loic Dachary

erasure code : plugin, interface and glossary documentation updates

- replace the erasure code plugin abstract interface with a doxygen link
  that will be populated when the header shows in master
- update the plugin documentation to reflect the current draft implementation
- fix broken link to PGBackend-h
- add a glossary to define chunk, stripe, shard and strip with a drawing

http://tracker.ceph.com/issues/4929 refs #4929

Signed-off-by: Loic Dachary <loic@dachary.org>

Revision bebba3c8 - 08/21/2013 04:09 PM - Loic Dachary

doc: fix erasure code formatting warnings and errors

http://tracker.ceph.com/issues/4929 refs #4929

Signed-off-by: Loic Dachary <loic@dachary.org>

Revision 157f2227 - 08/22/2013 03:45 PM - Loic Dachary

doc: fix erasure code formatting warnings and errors

http://tracker.ceph.com/issues/4929 refs #4929

Signed-off-by: Loic Dachary <loic@dachary.org>

Revision 14c31ddf - 08/27/2013 12:13 PM - Loic Dachary

doc : erasure code developer notes updates

- unify conventions to match those used by jerasure (data chunk = K,
  coding chunk = M, use coding instead of parity, use erasures instead
  of erased)

- make lines 80 characters long

- modify the descriptions to take into account that the chunk rank
  will encoded in the pool name and not on a per object basis

- remove the doxygen link to ErasureCodeInterface because it fails
  doc: asphyxiate does not support class
  http://tracker.ceph.com/issues/6115

- only systematic codes are considered at this point (all jerasure
  techniques are systematic). Although the API could be extended to
  include non systematic codes, it is probably a case of over
  engineering at this point.

- add link to
  http://tracker.ceph.com/issues/6113
  add ceph osd pool create [name] [key=value]

- update the plugin system description to match the proposed implementation http://tracker.ceph.com/issues/5877

http://tracker.ceph.com/issues/4929 refs #4929

Reviewed-by: Joao Eduardo Luis <joao.luis@inktank.com>

Signed-off-by: Loic Dachary <loic@dachary.org>

History

#1 - 05/29/2013 03:26 AM - Loic Dachary

maybe use erasure encoding from rozofs

#2 - 06/18/2013 07:09 AM - Loic Dachary

- Description updated

#3 - 06/20/2013 09:26 AM - Loic Dachary

- Description updated

#4 - 06/20/2013 10:51 AM - Loic Dachary

- Description updated

#5 - 06/20/2013 10:52 AM - Loic Dachary

- Description updated

#6 - 06/20/2013 10:53 AM - Loic Dachary

- Description updated

#7 - 06/20/2013 12:58 PM - Loic Dachary

the pad is only archived for so long, keep a pad backup

```
 Erasure encoded placement group / pool

    * Factor reusable components out of PG/ReplicatedPG and have PG/ReplicatedPG and ErasureCodedPG share only
 those components and a common PG API.
        * Advantages:
            * We constrain the PG implementations less while still allowing reuse some of the common logic.
            * Individual components can be tested without needing to instantiate an entire PG.
            * We will realize benefits from better testing as each component is factored out independent of im
plementing ErasureCodedPG.
        * Some possible common components:
            * Peering State Machine:  Currently, this is tightly coupled with the PG class.  Instead, it becom
es a seperate component responsible for orchestrating the peering process with a PG implementation via the PG
interface.  This would allow us to test specific behavior without creating an OSD or a PG.
            * ObjectContexts, object context tracking: this probably includes tracking read/write lock trackin
g for objects
            * Repop state?: not sure about this one, might be too different to generalize between ReplicatedPG
 and ErasureCodedPG
            * PG logs, PG missing: The logic for merging an authoritative PG log with another PG log while fil
ling in the missing set would benefit massively from being testable seperately from a PG instance.  It's possi
ble that the stripes involved in ErasureCodedPG will make this impractical to generalize.
    * To isolates ceph from the actual library being used ( jerasure, zfec, fecpp, ... ), a wrapper is impleme
nted. Each block is encoded into k data blocks and m parity blocks
        * context(k, m, reed-solomon|...) => context* c
        * encode(context* c, void* data) => void* chunks[k+m]
        * decode(context* c, void* chunk[k+m], int* indices_of_erased_chunks) => void* data // erased chunks a
```

```
re not used
        * repair(context* c, void* chunk[k+m], int* indices_of_erased_chunks) => void* chunks[k+m] // erased c
hunks are rebuilt
    * The ErasureEncodePG configuration is set to encode each object into k data objects and m parity objects.

        * It uses the parity ('INDEP') crush mode so that placement is intelligent. The indep  placement avoid
s moving around a shard between ranks, because a mapping  of [0,1,2,3,4] will change to [0,6,2,3,4] (or someth
ing) if osd.1 fails  and the shards on 2,3,4 won't need to be copied around.
        * The ErasureEncodedPG uses k + m OSDs
        * Each object is a chunk
        * The rank of the chunk is stored in the object attribute
        * Each chunk is divided into B bytes long parts coded independantly. For instance a 1GB chunk can be d
ivided in 4MB parts such that bytes 4BM to 8MB of each chunk can be processed independantly of the rest of the
 chunk. The K+M parts of each chunk that reside at the same position in the chunk are called a stripe.
    * ErasureEncodedPG implementation
        * Write a new object that does not need to be divided into parts because it is not too big ( 4MB for i
nstance )
            * encode(context* c, void* data) => void* chunks[k+m]
            * write chunk[i] to OSD[i] and set the "chunk_rank" attribute to i
        * Write offset, length on an existing chunk that is made of a number of independant parts coded separa
tely
            * read the stripes containing offset, length
            * map the rank of each chunk with the OSD on which it is stored
            * for each stripe, decode(context* c, void* chunk[k+m], int* indices_of_erased_chunks) => void* da
ta and append to a bufferlist
            * modify the bufferlist according to the write request, overriding the content that has been decod
ed with the content given in argument to the write
            * encode(context* c, void* data) => void* chunks[k+m]
            * write chunk[i] to the corresponding OSD[j]
        * Read offset, length
            * read the stripes containing [offset, offset + length]
            * for each stripe, decode(context* c, void* chunk[k+m], int* indices_of_erased_chunks) => void* da
ta and append to a bufferlist
        * Object attributes
            * duplicate the object attributes on each OSD
        * Scrubbing
            * for each object, read each stripe and write back the repaired part if necessary
        * Repair
            * When an OSD is decommissioned, when another OSD replaces it, for each object contained in a Eras
ureEncodedPG using this OSD, read the object, repair each stripe and write back the part that resides on the n
ew OSD
    * SJ - interface
        * Do we want to restrict the librados writes to just write full?  For writes, write full can be implem
ented much more efficiently than partial writes (no need to read stripes).
        * xattr can probably be handled by simply replicating across stripes.
        * omap options:
            * disable
            * erasure code??
            * replicate across all stripes - good enough for applications using omap only for limited metadata
        * How do we handle object classes?  A read might require a round trip to replicas to fulfill, we proba
bly don't want to block in the object class code during that time.  Perhaps we only allow reads from xattrs an
d omap entries from the object class?
    * SJ - random stuff
        * PG temp mappings need to be able to specify a primary independently of the acting set order (stripe
assignment, really).  This is necessary to handle backfilling a new acting[0].
        * An osd might have two stripes of the same PG due to a history as below.  This could be handled by al
lowing independent PG objects representing each stripe to coexist on the same OSD.
            * [0,3,6]
            * [1,3,6]
            * [9,3,0]
        * hobject_t and associated encodings/stringifications needs a stripe field
        * OSD map needs to track stripe as well as pg_t
        * split is straightforward -- yay
        * changing m,n is not easy

Use cases:
1. write full object
2. append to existing object?
3. pluggable algorithm
4. single-dc store (lower redundancy overhead)
5. geo-distributed store (better durability)

Questions:
     object stripe unit size.. per-object or per-pool?  => may as well be per-object, maybe with a pool (or ag
lorithm) default?
```

```
Work items:
    clean up OSD -> pg interface
    factor out common PG pieces (obc tracking, pg log handling, etc.)
    ...
    profit!
```

#8 - 07/01/2013 02:53 PM - Loic Dachary

- Description updated

#9 - 07/01/2013 04:05 PM - Loic Dachary

- Description updated

#10 - 07/08/2013 02:09 AM - Loic Dachary

- Description updated

#11 - 07/27/2013 12:13 PM - Loic Dachary

- Description updated

#12 - 08/01/2013 04:58 PM - Loic Dachary

- Description updated

#13 - 08/02/2013 07:05 PM - Samuel Just

- Description updated

#14 - 08/05/2013 06:46 AM - Loic Dachary

- Description updated

#15 - 08/07/2013 02:27 AM - Loic Dachary

- Description updated

#16 - 08/07/2013 09:37 AM - Loic Dachary

Archive of the erasure code session PAD of the Emperor session summit.

```
~5 minutes
    * short introduction : erasure code vs replica
    * erasure coding is a property of a pool and as such implemented in placement groups
    * summary of what has happened during Dumpling
        * limit operations to read / write / append : enough when you have automatic tiering
        * high level design of the implementation : PGBackend
        * erasure code library abstract API
        * factor out some PG components common to erasure code
    * what's going to happen for Emperor
        * completing the high level design
        * implement the OSD core changes and PG rearchitecture to introduce the PGBackend interface
        * prepare up to five alpha tester platforms. Who's interested ? mikedawson, joelio
        * ???

~20 minutes
    * high level design
        * walkthru : https://github.com/ceph/ceph/blob/wip-erasure-coding-doc/doc/dev/osd_internals/erasure_co
ding.rst
            * recovery using logs alone
            * delay deletion; version objects
        * question : coll_t to store chunk_id ? http://marc.info/?l=ceph-devel&m=137570617209017&w=4
            * yes: osd should be able to store different shards of the pg at the same time
            * add new cpg_t or pgshard_t (pool, seed, shard)
            * one PG instance -> 1 or more pgshard_t/coll_t's
        * question : what about snapshots ? http://marc.info/?l=ceph-devel&m=137569854906474&w=4
            * yes, could be... we can roll back
            * maybe not initially
        * question : what about watchers ? http://marc.info/?l=ceph-devel&m=137569854906474&w=4
            * probably not bother,
            * but could be.. it's backed by the object_info_t, which is just an attr and is logged is its enti
rety
        * question : what about PGBackend + PGBackendInterface ? http://marc.info/?l=ceph-devel&m=137569854906
474&w=4
            * PGBackend will be abstract
        * question : what's the story behind INDEP crush mode ?
            * prevent crush changing the position of osds in the result list
            * need to make a small change so that it can leave holes in the result (instead of shifting left)
        * question : is it reasonable to have the attribute key/value in the PG logs ?
            * consensus: yes; they are small, and it makes roll forward/back easy!
    * testing
        * write as much unit tests as humanly possible
        * add to integration tests to teuthology qa-suite
        * make it easier to install / run teuthology and upgrade as it evolves

~5 minutes
    * get involved
        * the design document splits the problem into smaller tasks that can be worked on relatively independa
ntly
        * the erasure coding plugin API is the easiest most independant task
        * top level URL http://tracker.ceph.com/issues/4929 Erasure encoded placement group
        * http://wiki.ceph.com/01Planning/02Blueprints/Emperor/Erasure_coded_storage_backend_%28step_2%29#Codi
ng_tasks
        * also linked from the design document https://github.com/ceph/ceph/blob/wip-erasure-coding-doc/doc/de
v/osd_internals/erasure_coding.rst
    * people involved
        * Samuel Just design and code review and refactoring
        * Loic Dachary development (full time)
        * ???

Notes:
    For erasure coding, rolling forward would not be necessary.

"chunk" -> "shard"?

Questions from IRC:
roll forward instead of roll back?
   -> the problem is that you can only roll forward if you have M/N of the shards

how efficient for small objects?
   -> not very?
   -> we pad out the stripe with 0's to do the erasure coding
   -> can use different stripe size for any object, but
   -> any write touches all OSDs in the PG, and small write -> ~13 ios is not ideal

Erasure coded RBD?
    Mutation is not efficient in erasure coded PG.
    Tiering would be another approach to that problem.
```

#17 - 08/21/2013 03:35 AM - Loic Dachary

- Description updated

#18 - 10/10/2013 03:28 AM - Loic Dachary

- Description updated

#19 - 10/11/2013 08:44 AM - Loic Dachary

- Description updated

#20 - 10/16/2013 11:11 AM - Loic Dachary

- Description updated

#21 - 10/16/2013 11:25 AM - Loic Dachary

- Description updated

#22 - 12/02/2013 05:03 AM - Loic Dachary

- Description updated

Add links to the Firefly summit to the description

#23 - 01/26/2014 02:23 AM - Loic Dachary

- Description updated

#24 - 02/06/2014 12:52 AM - Loic Dachary

- Description updated

#25 - 03/07/2014 05:01 AM - Loic Dachary

- Status changed from In Progress to Resolved