Supporting XRootD Vector Read in the Ceph Plugin

Context

Analysis jobs use ROOT Direct IO and the Vector Read call to access data stored in ECHO.
The Ceph plugin does not currently support Vector Read, so XRootD Cache needs to issue read calls for large amounts of data.
16 MB blocks are requested by the XRootD cache, leading to “read amplification”.
The XRootD cache processes on the worker nodes see time-outs from these read calls because the Ceph plugin cannot process requests for such large amounts of data..

Goal

Support Vector Read directly in the Ceph plugin. This will mean that the XRootD cache process will be able to call the dedicated Vector Read function instead of issuing “normal” Read requests. This should lead to better performance and no/reduce time-outs.

Include XRootD OSS / OFS layer diagram. Mention where plugins (implementation) fit into this.

Requirements

Correct Operation:

1. The Ceph plugin must correctly implement the Vector Read call (honouring the API from XRootD OSS layer to the implementation layer in the Ceph plugin).
2. The Ceph plugin must return the data requested correctly. (Redundant?)

Performance:

1. The Ceph plugin must access data efficiently.
2. The Ceph plugin must not cause timeouts between client and server

Resources

Code examples implementing Vector Read for POSIX and Cache backends in the XRootD server.

Advice from XRootD developers Andy Hanuschevsky and Brian Bockelman.
Experience understanding how RADOS Striper reads data in units of ‘stripe width’ (e.g. 4 KB for CMS, other pools may/will be different).
Source code for Ceph plugin, RADOS Striper, and RADOS libraries.

Approach

We anticipate the need for a test framework to prove correctness of code we develop. Look into the test suites that are in the XRootD server repository. Ideally, adapt the Vector Read client-side test into something that can use the Ceph plugin directly to provide a unit test, before moving onto end-to-end tests (from client to server), and integration tests (e.g. ROOT programs us

Appendix 1: Example ReadV implementation in XrdOss layer

The code below is the “default” implementation of Vector Read for a file storage backend. The loop calls a “read from offset” method `Read(data, offset, size)`

```
/******************************************************************/
/*                             R e a d V                          */
/******************************************************************/

ssize_t XrdOssDF::ReadV(XrdOucIOVec *readV,
                        int          n)
{
   ssize_t nbytes = 0, curCount = 0;
   for (int i=0; i<n; i++)
```

```
        {curCount = Read((void *)readV[i].data,
                         (off_t)readV[i].offset,
                        (size_t)readV[i].size);
         if (curCount != readV[i].size)
            {if (curCount < 0) return curCount;
             return -ESPIPE;
            }
         nbytes += curCount;
        }
   return nbytes;
}
```

The “read from offset” method in the XrdOSS layer calls an implementation layer method in the appropriate plugin. For the XRootD Ceph plugin, “read from offset” is matched by the XrdCephPosix method:

```
ssize_t ceph_posix_pread(int fd, void *buf, size_t count, off64_t
offset)
```