Adding fallocate() API support

Swapnil Pimpale

December 9, 2013

1. Requirements

We need to implement an fallocate() method for llite, and transport this to the OSTs to interface with the underlying OSD's fallocate() code (for ldiskfs, ZFS has no such method)

2. Use cases

The fallocate() API has been added to the 2.6.24 version of the Linux kernel to provide both persistent space reservation (block preallocation for a file, possibly beyond the file size, without having to write zeros to the whole file) and for the reverse operation of hole punching (freeing allocated blocks in the middle or end of a file). Being able to preallocate space for a file is very useful for HPC applications that know the size of the output file in advance, and helps make better allocation decisions based on the file size.

3. Functional Specification

3.1. New ioctl method of cl_object_operations{}

A new operation coo_ioctl() will be added into cl_object_operations{} which will be used to support fallocate(2) and other operations in the future.
cl_object_ioctl() will be defined to call coo_ioctl() of each layer.

3.2. ll_file_fallocate()

This will be the callback of inode's fallocate() method. ll_file_fallocate() will initiate a CIT_MISC IO and then invoke cl_object_ioctl() to call down to LOV and OSC Layer.

3.3. SLP, VVP layers:

There will be no implementation at SLP and VVP layers for fallocate() API.

3.4. LOV Layer

At this layer, we will calculate the fallocated region by stripe data and then call coo_ioctl() of the underlying (OSC) layer.

HSM-released file will also be handled at this layer. If the file is released, it will have to be restored before it's able to continue fallocate REQ. Otherwise there will not be any objects corresponding to this file and fallocate will fail.
To get this done, an intent will be added to cl_io{} to identify that the IO is for fallocate() as follows:

```
stuct cl_io {
                ...
                /* how many pages were written/discarded /
                unsigned int      fi_nr_written;
        } ci_fsync;
        /* newly added field /
        struct cl_misc_io {
                int mi_intent; /* the intent of this MISC IO, for fallocate() right now */
        } ci_misc;
};
#define CI_MISC_FALLOC 0x1
```

Thus, in lov_io_init_released() we can check if the MISC IO is for fallocate() then restore the file i.e., to set ci_restore_needed and return to the upper layer.

3.5. OSC Layer

A new RPC type OST_FALLOC will be introduced for fallocate(). The RPC will not modify the file's a/c/m_time if KEEP_SIZE is set.
The exact extent should be aligned to the chunksize of the OSC.

FALLOC_FL_PUNCH_HOLE functionality of fallocate() will use the existing OST_PUNCH RPC. FALLOC_FL_PUNCH_HOLE is not supported in 2.6.32-358 kernels but we will support it from the client side.

3.6. OSD Layer

fallocate() for Lustre will support ldiskfs-osd. It will not support ZFS-OSD and will have limited support for page_mkwrite. fallocate() to reserve space for ZFS will return -EOPNOTSUPP

PUNCH and truncate code will be identical, with truncate being a special case of punch to ~0UL

If fallocated file is being overwritten then the grant will be returned to the same client on every write (ofd/ofd_grant.c)

3.7. Recovery

OST_FALLOC will have a 'transno' so that it can be replayed during recovery.

3.8. Grants

Calling fallocate() will be treated exactly the same as a write – it consumes space on the OST, so it will consume grant as well. This will be similar to overwriting a file after a normal write.

PUNCH will not consume any grants. PUNCH is basically truncate in the middle of a file which should drop any cached pages on the client and on the OST just like a truncate does.

3.9. Truncate

Truncate will release any resource reserved by allocation REQ beyond the truncated size.

3.10. Write vs Rewrite

No difference in ldiskfs

4. Focus for inspection

- Do we need to initiate a local deallocation REQ to release the space allocated by the evicted client?