

Adding fallocate() API support

Swapnil Pimpale

January 7, 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. Synopsis

```
int fallocate(int fd, int mode, off_t offset, off_t len);
```

4. Functional Specification

4.1. SLP, VVP Layers

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

4.2. ll_fallocate()

This will be the callback of inode's fallocate() method. Based on the mode passed, it will either call cl_falloc_prealloc() or cl_falloc_punch().

4.3. fallocate – Prealloc (default mode)

4.3.1. Llite layer: cl_falloc_prealloc()

Prealloc operation will be multiplexed with CIT_SETATTR. The following new fields will be added to ci_setattr{}

```
struct cl_setattr_io {  
    ...  
    int    sa_falloc_mode;  
    loff_t sa_falloc_offset;  
    loff_t sa_falloc_len;  
} ci_setattr;
```

These values will be passed down the layers finally to the OSD's fallocate() method.

cl_falloc_prealloc() will get CAPA_OPC_OSS_WRITE capability, initiate a CIT_SETATTR IO and then call cl_io_loop().

Prealloc on an HSM released file will be treated similar to a write operation i.e., the released file will be restored with ll_layout_restore() before proceeding to the prealloc operation.

4.3.2. LOV layer:

At this layer, the fallocated region will be calculated by stripe data.

4.3.3. OSC layer:

A new RPC type OST_PREALLOC will be introduced for fallocate(). This RPC will not

modify the file's size if `FALLOC_FL_KEEP_SIZE` is set even if `offset+len` is greater than the file size. Preallocating zeroed blocks beyond the end of file in this manner is useful for optimizing append workloads.

4.3.4. 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`.

4.3.5. Grants:

`fallocate()` will bypass grants on the client side. The 'write' after `fallocate()` will be treated as a normal 'write' when dealing with grants. The 'write' will be guaranteed not to fail with `-ENOSPC` because the disk space is pre-allocated on the OST.

4.4. `fallocate` – `FALLOC_FL_PUNCH_HOLE` | `FALLOC_FL_KEEP_SIZE`

4.4.1. Llite layer: `cl_falloc_punch()`

Similar to `cl_falloc_prealloc()`. Even in this case, an HSM released file will be restored before performing a punch operation.

4.4.2. LOV layer:

The punch region will be calculated by stripe data at this layer.

4.4.3. OSC layer:

`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 kernel but it will be supported from the client side.

4.4.4. OSD layer:

Punch and truncate code will be identical, with truncate being a special case of punch to `~0UL`. `ofd_punch_hdl()`, `ofd_object_punch()`, `osd_punch()` will be modified to support punch in the middle of the file. This code path will end with a call to `ldiskfs_fallocate()`.

4.4.5. Grants:

`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.

4.5. `fallocate` – `FALLOC_FL_COLLAPSE_RANGE`

This new `fallocate` mode will be added to the upstream `fallocate()` soon. The semantics of this flag as mentioned in the patch series are as follows:

- i) It collapses the range lying between offset and length by removing any data blocks which are present in this range and then updates all the logical offsets of the extents beyond “offset + len” to nullify the hole created. Thus, it does not leave a hole.
- ii) It should be used exclusively. No other flag in combination.
- iii) Offset and length supplied to `fallocate()` should be FS block size aligned.

A major issue with this mode is if the file has multiple stripes and the collapse operation is not a multiple of (`stripe_size * stripe_count`) in size. That would mean data needs to be moved between OSTs. So, the multi-stripe case would return `-EOPNOTSUPP` if the collapse range is not well aligned.

4.6. Truncate:

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

4.7. Recovery:

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

5. Focus for inspection

- The restoring of an HSM released file before a prealloc operation is based on the assumption that, prealloc will be followed by a write.