DLD Size-on-MDS LLOG'ing.

Vitaly Fertman

2007-05-04

1 Introduction.

Size-on-MDS metadata improvement includes the caching of the inode size, blocks, ctime and mtime on the MDS. Such an attribute caching allows clients to avoid making RPCs to the OSTs to find the attributes encoded in the file objects kept on those OSTs what results in the significantly improved performance of listing directories.

The current DLD involves LLOG'ing Size-on-MDS attribute changes on OST, what guarantees that OST object attribute changes will not be lost somewhere on the way to MDS (do to a node failure) and will allow MDS to re-enable Size-on-MDS caching for the inode with the proper attributes. The recovery itself from a node failure is out of scope of the current document.

2 Requirements.

2.1 Architectural requirement.

An IO epoch mechanism is introduced that allows for two attribute retrieval mechanisms:

- OST based when files are active for IO
- MDS based when files are not active for IO

Broadly speaking an IO epoch for a file starts when a file is opened for writing and closes when (i) the file is closed (ii) the IO caches have been flushed. Attribute change may only happen when file is active for IO, i.e. within some IO epoch.

2.2 Functional requirements

- Client identifies its IO activity by an IO epoch identifier (number).
- OST creates transactionally one llog record for every IO epoch of every object that has a Size-on-MDS attribute change on a persistent store (disk).
- OST propagates llog cookie in reply to IO operation and to GETATTR (which identifies a particular epoch) requests.
- Client propagates llog cookies to MDS along with the Size-on-MDS attribute update when IO epoch is closed on the client.
- Client obtains Size-on-MDS attributes along with llog cookies for a particular epoch from OSTs and propagates this info to MDS, when MDS explicitly asks client to provide a Size-on-MDS attribute update.
- MDS sends llog cancel requests to OSTs for an IO epoch when no IO epoch is opened on the inode, and when Size-on-MDS attributes are obtained and committed on a persistent store (disk).
- The log record must exists until the cancellation or the destroy request comes on the OST.

3 Definitions.

IO epoch. The inode on the MDS is in the IO epoch if somewhere in the cluster an IO can be initiated through interaction with OST's.

IO epoch number. The number that uniquely identifies the epoch an inode is in.

4 Use case scenarios.

4.1 Client truncates a file.

- LLITE prepares and sends SETATTR RPC on MDS to check the permissions.
- MDT checks permissions and opens an IO epoch on the inode.
- LLITE prepares and sends PUNCH RPC to every OST which keeps a file object, IO epoch number is sent here.
- OST truncates object, checks if there is llog record for this epoch of this object, and creates a new llog record if not, IO epoch number is llog'ed. Llog cookie is sent back in reply.

- LLITE prepares and sends DONE_WRITING RPC to the MDS, packing there SOM attributes along with llog cookies.
- MDT closes the IO epoch and updates inode attributes if there is no more IO Epoch holder and attributes are valid. Otherwise, MDS waits for the last IO epoch holder to close the IO epoch or asks the client for a SOM attribute update for the closed epoch. If MDS updates attributes, it waits for the commit and sends LLOG_CANCEL RPC of OST which llog cookies are obtained for the closed IO epoch.

4.2 Client writes to a file.

- LLITE prepares and sends ENQUEUE OPEN RPC to open a file for write.
- MDT opens an IO epoch on the file and sends it back in reply.
- LLITE saves IO epoch in the inode.
- LLITE writes to the file, when pages are sent to OST, it prepares and sends WRITE RPC to OSTs and the IO Epoch number the inode is in is sent along in this RPC.
- OST writes data, checks if there is llog record for this epoch of this object, and creates a new llog record if not, IO epoch number is llog'ed. Llog cookie is sent back in reply.
- LLITE stores llog cookies in the inode and keeps them there until IO Epoch is getting closed on this client.
- LLITE prepares and sends CLOSE RPC to MDS. If there is a dirty cache by this time, DONE_WRITING RPC which will close IO Epoch will be sent later. Otherwise, CLOSE will also close IO epoch. If clients has all the locks over all the stripes when it closes IO Epoch, it packs object attributes along with the llog cookies from the inode to the request to MDS.
- MDT closes file. If IO Epoch is closed too, MDT updates inode attributes if there is no more IO Epoch holder and attributes are valid. Otherwise MDS waits for the last IO epoch holder to close the epoch or asks the client for a SOM attribute update for the closed epoch. If MDS updates attributes, it waits for the commit and sends LLOG_CANCEL RPC to OST which llog cookies are obtained for the closed IO epoch.

4.3 SOM Attribute Update.

- MDT asks the last IO epoch holder (client) for the Size-on-MDS attribute update when it closes IO Epoch on CLOSE or DONE_WRITING RPC with no attribute update or it was not the only IO Epoch holder.

- LLITE prepares and sends GETATTR RPC to OSTs for all the file objects regardless the objects extent locks it has, identifying the IO Epoch it needs Size-on-MDS attributes for – the IO Epoch that has been closed, not the current one on the inode.
- OST packs to reply the current attributes (not for a particular epoch) and the llog cookie for the specified IO epoch.
- LLITE sends obtained attributes and llog cookies to MDS.
- MDT updates obtained attributes if they are of the last IO Epoch (no new epoch has been opened).
- MDS waits for the attribute commit, if it updates them, and sends LLOG_CANCEL RPCs to OST from which llog cookies has been obtained from in this IO epoch attribute update. Otherwise, if attributes are not updated, MDS just sends LLOG_CANCEL RPCs.

5 Functional specification.

5.1 Data definitions.

1. Keep llog cookies in the inode while IO Epoch is opened.

   All cookies are kept in one memory chunk for the easier handling, in lov_stripe_md data.

   ```
   struct lov_stripe_md {
       ...
       struct llog_cookie *lsm_cookies;
       ...
   };
   ```

2. OSC asynchronous page gets the pointer to the llog_cookie in lov_stripe_md.

   ```
   struct OSC_async_page {
       ...
       struct llog_cookie      *oap_cookie;
   };
   ```

3. Pass llog cookies to MDC through md_op_data.

   ```
   struct md_op_data {
       ...
       __u32 op_numcookies;
       struct llog_cookie *op_cookies;
       ...
   };
   ```

4. Keep received llog cookies in mdt_info_thread in MDT.

```
struct mdt_thread_info {
    ...
    struct llog_cookie        *mti_cookie;
    int                        mti_cookie_size;
};
```

5. Keep received llog cookies and lov_mds_md in mdt_txn_info for the postponed handling.

```
struct mdt_txn_info {
    ...
    struct llog_cookie *txi_cookies;
    struct lov_mds_md  *txi_lmm;
    int                 txi_lmm_size;
};
```

6. Keep cookie in the inode on OST.

```
struct ost_filterdata {
    ...
    struct llog_cookie ofd_cookie;
};
```

7. LLOG OST object id and group, it helps to find the corresponding inode by these fields. Do not llog mfd_fid anymore.

```
struct llog_size_change_rec {
    ...
    __u64 lsc_id;
    __u64 lsc_gr;
};
```

5.2 MD Interface.

1. int (*m_setattr)(struct obd_export *, struct md_op_data *, void *, int, struct ptlrpc_request **);

   Cookies are passed through md_op_data instead of separate ea2 & ea2len attributes to m_setattr.

5.3 OBD Interface.

1. int (*o_prep_async_page)(struct obd_export *exp, struct lov_stripe_md *lsm, int pos, struct page *page, obd_off offset, struct obd_async_page_ops *ops, void *data, void **res);

Specify the stripe number @pos instead of the explicit lov_oinfo pointer in o_prep_async_page(). See osc_prep_async_page() for details.

5.4 MD Device Interface.

1. int (*mdo_llog_cancel)(const struct lu_env *env, struct md_device *m, struct lov_mds_md *ma_lmm, int ma_lmm_size, struct llog_cookie *cookies);

   Pass Size-on-MDS cookies down from MDT to MDS.

```
static struct md_device_operations cmm_md_ops = {
    ...
    .mdo_llog_cancel     = cmm_llog_cancel,
};
struct md_device_operations mdd_ops = {
    ...
    .mdo_llog_cancel    = mdd_llog_cancel,
};
```

6 Logic specification.

6.1 OBDO.

1. void md_from_obdo(struct md_op_data *op_data, struct obdo *oa, int count, obd_flag valid)

   Copy IO Epoch number from @oa to @op_data if OBD_MD_FLEPOCH is valid, and @count cookies from @oa if OBD_MD_FLCOOKIE is valid.

```
if (valid & OBD_MD_FLEPOCH) {
    op_data->op_ioepoch = oa->o_easize;
    op_data->op_flags |= MF_SOM_CHANGE;
    if (oa->o_valid & OBD_MD_FLCOOKIE) {
        op_data->op_numcookies = count;
        op_data->op_cookies = obdo_logcookie(oa);
    }
}
```

6.2 LLITE.

1. void ll_epoch_open(struct ll_inode_info *lli, int ioepoch)

   Set up the opened IO Epoch number to lli and zero llog cookies it new epoch starts.

```
if (ioepoch && lli->lli_ioepoch != ioepoch) {
    memset(lli->lli_smd->lsm_cookies, 0, sizeof(struct llog_cookie) *
           lli->lli_smd->lsm_stripe_count);
    lli->lli_ioepoch = ioepoch;
}
```

2. static int ll_och_fill(struct obd_export *md_exp, struct ll_inode_info *lli, struct lookup_intent *it, struct obd_client_handle *och)

   Call ll_epoch_open() to set ioepoch to the inode and clear old llog cookies.

3. int ll_setattr_raw(struct inode *inode, struct iattr *attr)

   Call ll_epoch_open() to set ioepoch to the inode and clear old llog cookies. It is needed for the later ll_truncate() that takes IO Epoch number from the inode.

4. void ll_truncate(struct inode *inode)

   Pass IO Epoch number taken from the inode to OSTs for the proper SOM attribute change llog'ing.

```
oa.o_easize = lli->lli_ioepoch;
oa.o_valid |= OBD_MD_FLEPOCH;
```

5. void ll_epoch_close(struct inode *inode, struct md_op_data *op_data, struct obd_client_handle **och, unsigned long flags)

   Install llog cookies into md_op_data when client informs MDS that Size-on-MDS attributes are changed (i.e. MF_SOM_CHANGE is set).

```
op_data->op_flags |= MF_SOM_CHANGE;
op_data->op_numcookies = lli->lli_smd->lsm_stripe_count;
op_data->op_cookies = lli->lli_smd->lsm_cookies;
```

6. static int ll_setattr_done_writing(struct inode *inode, struct md_op_data *op_data)

   Install llog cookies into md_op_data (MF_SOM_CHANGE is always set here). Also send SOM attributes if client has all extent locks.

```
op_data->op_numcookies = lli->lli_smd->lsm_stripe_count;
op_data->op_cookies = lli->lli_smd->lsm_cookies;
if (!ll_local_size(inode)) {
    /* Send Size-on-MDS Attributes if valid. */
    op_data->op_attr.ia_valid |= ATTR_MTIME_SET | ATTR_CTIME_SET |
                                 ATTR_SIZE | ATTR_BLOCKS;
}
```

7. int ll_inode_getattr(struct inode *inode, struct obdo *obdo)

   Do not erase given @obdo->o_valid flags, some data may be valid, i.e. IO Epoch number we want the attributes for. Do not erase Size-on-MDS related flags when exiting as well.

```
oinfo.oi_oa->o_valid |= ... ;
...
oinfo.oi_oa->o_valid &= (... | OBD_MD_FLEPOCH | OBD_MD_FLCOOKIE);
```

8. int ll_sizeonmds_update(struct inode *inode, struct lustre_handle *fh, __u64 ioepoch)

   Obtain object attributes for a particular epoch (probably not the currently opened on the inode) and pass them to MDS. Call ll_inode_getattr() for the wanted epoch and pass the results to MDS through ll_md_setattr().

```
count = lli->lli_lsm->lsm_stripe_count - 1;
count = sizeof(*oa) + count * sizeof(struct llog_cookie);
OBD_ALLOC(oa, count);
...
oa->o_valid |= OBD_MD_FLEPOCH;
oa->o_easize = ioepoch;
...
rc = ll_inode_getattr(inode, oa);
...
md_from_obdo(op_data, oa, lli->lli_lsm->lsm_stripe_count, oa->o_valid);
...
rc = ll_md_setattr(inode, op_data);
OBD_FREE(oa, count);
```

6.3 LOV.

1. Allocation/deallocation memory for the llog cookies in lov_stripe_md

```
int lov_alloc_memmd(struct lov_stripe_md **lsmp, int stripe_count,
                    int pattern, int magic)
{
    ...
    OBD_ALLOC((*lsmp)->lsm_cookie,
              sizeof(struct llog_cookie) * stripe_count);
    ...
}
void lov_free_memmd(struct lov_stripe_md **lsmp)
{
    ...
```

```
    OBD_FREE(lsm->lsm_cookie,
             sizeof(struct llog_cookie) * lsm->lsm_stripe_count);
    ...
}
```

2. static int lov_llog_repl_cancel(struct llog_ctxt *ctxt, struct lov_stripe_md *lsm, int count, struct llog_cookie *cookies, int flags)

   Send cancel to those OST only that provided a valid cookie.

```
if (cookies->lgc_lgl.lgl_oid == 0)
    continue;
```

3. static void lov_common_cookie_unpack(struct lov_request_set *set)

   Copy obtained cookies into the lov_stripe_md @set->set_oi->oi_md.

```
static void lov_common_cookie_unpack(struct lov_request_set *set)
{
    struct list_head *pos;
    ENTRY;
    LASSERT(set);
    list_for_each(pos, &set->set_list) {
        struct lov_request *req = list_entry(pos, struct lov_request,
                                             rq_link);
        /* Move obtained cookies to lsm */
        if (req->rq_oi.oi_oa->o_valid & OBD_MD_FLCOOKIE) {
            memcpy(set->set_oi->oi_md->lsm_cookies + req->rq_stripe,
                   obdo_logcookie(req->rq_oi.oi_oa),
                   sizeof(struct llog_cookie));
        }
    }
}
```

4. static void lov_getattr_cookie_unpack(struct lov_request_set *set)

   Copy obtained cookies into the given obdo @set->set_oi->oi_oa.

```
static void lov_common_cookie_unpack(struct lov_request_set *set)
{
    struct list_head *pos;
    ENTRY;
    LASSERT(set);
    list_for_each(pos, &set->set_list) {
        struct lov_request *req = list_entry(pos, struct lov_request,
                                             rq_link);
        /* Move obtained cookies to obdo */
```

```
            if (req->rq_oi.oi_oa->o_valid & OBD_MD_FLCOOKIE) {
                struct llog_cookie *lc;
                lc = obdo_logcookie(set->set_oi->oi_oa) + req->rq_stripe;
                memcpy(lc, obdo_logcookie(req->rq_oi.oi_oa),
                       sizeof(struct llog_cookie));
            }
        }
    }
```

5. int lov_fini_brw_set(struct lov_request_set *set)

   Gather obtained cookies into lov_stripe_info @set->set_oi->oi_md.

```
if (set->set_completes) {
    ...
    lov_common_cookie_unpack(set);
}
```

6. int lov_fini_punch_set(struct lov_request_set *set)

   Gather obtained cookies into lov_stripe_info @set->set_oi->oi_md.

```
if (set->set_completes) {
    if (!set->set_success)
        rc = -EIO;
    else
        lov_common_cookie_unpack(set);
}
```

7. int lov_fini_getattr_set(struct lov_request_set *set)

   Gather obtained cookies into given obdo @set->set_oi->oi_oa.

```
if (set->set_completes) {
    ...
    lov_getattr_cookie_unpack(set);
}
```

6.4 OSC.

1. int osc_prep_async_page(struct obd_export *exp, struct lov_stripe_md *lsm, int pos, cfs_page_t *page, obd_off offset, struct obd_async_page_ops *ops, void *data, void **res)

   Set up osc_async_page to point to the correct llog_cookie in lov_stripe_md. Get lov_oinfo by the stripe number instead of the explicit pointer.

```
oap->oap_cookie = &lsm->lsm_cookies[pos];
oap->oap_loi = &lsm->lsm_oinfo[pos];
```

2. static void osc_ap_completion(struct client_obd *cli, struct obdo *oa, struct osc_async_page *oap, int sent, int rc)

   When IO completes, copy llog cookie to lsm, it the cookie is provided by OST.

```
if (rc == 0 && oa != NULL) {
    if (oa->o_valid & OBD_MD_FLCOOKIE)
        memcpy(oap->oap_cookie, obdo_logcookie(oa),
               sizeof(*oap->oap_cookie));
}
```

3. static int osc_llog_init(struct obd_device *obd, struct obd_llogs *llogs, struct obd_device *tgt, int count, struct llog_catid *catid, struct obd_uuid *uuid)

   Install the replicator methods into osc_size_repl_logops.

```
if (osc_size_repl_logops.lop_cancel != llog_obd_repl_cancel) {
    osc_size_repl_logops = llog_client_ops;
    osc_size_repl_logops.lop_cancel = llog_obd_repl_cancel;
    osc_size_repl_logops.lop_connect = llog_repl_connect;
    osc_size_repl_logops.lop_sync = llog_obd_repl_sync;
}
```

6.5 OST.

1. static int ost_getattr(struct obd_export *exp, struct ptlrpc_request *req)

   Send back llog cookie in the reply if obd_getattr() has packed it into @oinfo->oi_oa.

```
if (obdo_logcookie(oinfo.oi_oa)->lgc_lgl.lgl_oid)
    oinfo.oi_oa->o_valid |= OBD_MD_FLCOOKIE;
```

2. static int ost_punch(struct obd_export *exp, struct ptlrpc_request *req, struct obd_trans_info *oti)

   Send back llog cookie in the reply if obd_punch() has packed it into @oti.

```
repbody->oa = *oinfo.oi_oa;
repbody->oa.o_valid &= ~OBD_MD_FLCOOKIE;
...
req->rq_status = obd_punch(exp, &oinfo, oti, NULL);
...
if (oti->oti_onecookie.lgc_lgl.lgl_oid) {
    repbody->oa.o_valid |= OBD_MD_FLCOOKIE;
    memcpy(obdo_logcookie(&repbody->oa), &oti->oti_onecookie,
```

```
                sizeof(oti->oti_onecookie));
}
```

3. static int ost_brw_write(struct ptlrpc_request *req, struct obd_trans_info *oti)

   Send back llog cookie in the reply, if obd_commitrw() has packed it into @oti.

```
rc = obd_commitrw(OBD_BRW_WRITE, req->rq_export, &repbody->oa,
                  objcount, ioo, npages, local_nb, oti, rc);
...
if (oti->oti_onecookie.lgc_lgl.lgl_oid) {
    repbody->oa.o_valid |= OBD_MD_FLCOOKIE;
    memcpy(obdo_logcookie(&repbody->oa), &oti->oti_onecookie,
           sizeof(oti->oti_onecookie));
}
```

4. int ost_llog_handle_cancel(struct ptlrpc_request *req)

   Find LLOG records for the received LLOG cookies, find the corresponding inodes and unpin them. Cancel LLOG records after that.

```
rc = filter_log_prep_cancel(req);
if (rc)
    RETURN(rc);
RETURN(llog_origin_handle_cancel(req));
```

5. int ost_handle(struct ptlrpc_request *req)

   Call ost_llog_handle_cancel() instead of llog_origin_handle_cancel() to unpin corresponding inodes before canceling LLOG records.

6.6 Filter.

1. static int filter_llog_init(struct obd_device *obd, struct obd_llogs *llogs, struct obd_device *tgt, int count, struct llog_catid *catid, struct obd_uuid *uuid)

   Install originator methods into filter_size_orig_logops.

```
filter_size_orig_logops = llog_lvfs_ops;
filter_size_orig_logops.lop_setup = llog_obd_origin_setup;
filter_size_orig_logops.lop_cleanup = llog_obd_origin_cleanup;
filter_size_orig_logops.lop_add = llog_obd_origin_add;
filter_size_orig_logops.lop_connect = llog_origin_connect;
```

2. void filter_epoch2cookie(struct inode *inode, __u32 ioepoch, + struct llog_cookie *cookie)

   If there is a llog cookie kept in the inode, copy it to @cookie.

```
struct ost_filterdata *ofd = inode->i_filterdata;
ENTRY;
LOCK_INODE_MUTEX(inode);
if (ofd && ofd->ofd_epoch <= ioepoch)
    memcpy(cookie, &ofd->ofd_cookie, sizeof(*cookie));
UNLOCK_INODE_MUTEX(inode);
EXIT;
```

3. static int filter_getattr(struct obd_export *exp, struct obd_info *oinfo)

   If OBD_MD_FLEPOCH flag is set in the request, return LLOG cookie for the specified epoch if LLOG record exists.

```
if (oinfo->oi_oa->o_valid & OBD_MD_FLEPOCH) {
    filter_epoch2cookie(dentry->d_inode, oinfo->oi_oa->o_easize,
                        obdo_logcookie(oinfo->oi_oa));
}
```

4. int filter_setattr_internal(struct obd_export *exp, struct dentry *dentry, struct obdo *oa, struct obd_trans_info *oti)

   If IO Epoch is specified by the client, this attribute change should be llog'ed. Return the cookie of the created LLOG record.

```
if (oa->o_valid & OBD_MD_FLEPOCH) {
    rc = filter_log_sz_change(exp->exp_obd, oa,
                              &oti->oti_onecookie, inode);
}
```

5. int filter_commitrw_write(struct obd_export *exp, struct obdo *oa, int objcount, struct obd_ioobj *obj, int niocount, struct niobuf_local *res, struct obd_trans_info *oti, int rc)

   LLog this attribute change and return the cookie of the created LLOG record.

```
rc = filter_log_sz_change(exp->exp_obd, oa,
                          &oti->oti_onecookie, inode);
```

6. int filter_log_sz_change(struct obd_device *obd, struct obdo *oa, struct llog_cookie *logcookie, struct inode *inode)

   There is no mds_fid nor ioepoch in the parameter list, they are taken from the given @oa instead, as well as id/group which are LLOG'ed instead of mds_fid now.

   Add a new llog record to the catalog. If there is a llog record already for the previous epoch, remove it. after adding a new one.

```
__u32 ioepoch = oa->o_easize;
LASSERT((oa->o_valid &
         (OBD_MD_FLID & OBD_MD_FLGROUP & OBD_MD_FLEPOCH)) ==
        (OBD_MD_FLID & OBD_MD_FLGROUP & OBD_MD_FLEPOCH));
...
if (!ofd) {
    OBD_ALLOC(ofd, sizeof(*ofd));
    if (!ofd) {
        UNLOCK_INODE_MUTEX(inode);
        RETURN(-ENOMEM);
    }
    igrab(inode);
    inode->i_filterdata = ofd;
}
old_epoch = ofd->ofd_epoch;
ofd->ofd_epoch = ioepoch;
/* the decision to write a record is now made, unlock */
UNLOCK_INODE_MUTEX(inode);
ctxt = llog_get_context(obd, LLOG_SIZE_ORIG_CTXT);
if (ctxt == NULL) {
    CWARN("llog subsys not setup for SIZE\n");
    RETURN(-ENOENT);
}
OBD_ALLOC(lsc, sizeof(*lsc));
if (lsc == NULL)
    RETURN(-ENOMEM);
lsc->lsc_hdr.lrh_len = lsc->lsc_tail.lrt_len = sizeof(*lsc);
lsc->lsc_hdr.lrh_type = OST_SZ_REC;
lsc->lsc_ioepoch = ioepoch;
lsc->lsc_id = oa->o_id;
lsc->lsc_gr = oa->o_gr;
/* Add new llog record. */
rc = llog_cat_add_rec(ctxt->loc_handle, &lsc->lsc_hdr,
                      logcookie, NULL);
/* Cancel old llog record. */
rc1 = llog_cat_cancel_records(ctxt->loc_handle, 1, &ofd->ofd_cookie);
if (rc1) {
    CWARN("Failed to cancel cookie for epoch=%u ino=%lu",
          old_epoch, inode->i_ino);
}
ofd->ofd_cookie = *logcookie;
OBD_FREE(lsc, sizeof(*lsc));
RETURN(rc);
```

7. int filter_log_prep_cancel(struct ptlrpc_request *req)

   Find LLOG records for the received cookies, find inodes pinned in the

memory for these records and unpin them.

```
struct llog_cookie *logcookies;
int num_cookies, rc = 0, i;
struct llog_ctxt *ctxt;
logcookies = lustre_msg_buf(req->rq_reqmsg, REQ_REC_OFF,
num_cookies = lustre_msg_buflen(req->rq_reqmsg, REQ_REC_OFF) /
              sizeof(*logcookies);
if (logcookies == NULL || num_cookies == 0) {
    DEBUG_REQ(D_HA, req, "no cookies sent");
    RETURN(-EFAULT);
}
/* Clean up inodes for SIZE_ORIG only. */
if (logcookies->lgc_subsys != LLOG_SIZE_ORIG_CTXT)
    RETURN(0);
ctxt = llog_get_context(req->rq_export->exp_obd,
                        logcookies->lgc_subsys);
if (ctxt == NULL) {
    CWARN("llog subsys not setup or already cleanup\n");
    RETURN(-ENOENT);
}
for (i = 0; i < num_cookies; i++, logcookies++) {
    struct llog_handle *cathandle = ctxt->loc_handle;
    struct llog_process_cat_data cdata;
    struct llog_cancel_data cd;
    LASSERT(cathandle != NULL);
    cdata.first_idx = logcookies->lgc_index - 1;
    cdata.last_idx  = logcookies->lgc_index;
    cd.cookie = logcookies;
    cd.obd    = req->rq_export->exp_obd;
    rc = llog_process(cathandle, filter_cancel_cb, &cd, &cdata);
    if (rc != 0 && rc != LLOG_PROC_BREAK)
        RETURN(rc);
}
RETURN(0);
```

8. static int filter_cancel_cb(struct llog_handle *llh, struct llog_rec_hdr *rec, void *data)

   Compare the current llog record with the wanted one. If matches, find the corresponding inode and unpin it.

```
struct llog_size_change_rec *lsc = (struct llog_size_change_rec *)rec;
struct llog_cancel_data *cd = (struct llog_cancel_data *)data;
struct dentry *dentry;
if (memcmp(&cd->cookie->lgc_lgl, &llh->lgh_id, sizeof(llh->lgh_id)) ||     cd->cook
```

```
    return 0;
/* Record matches. Find and handle inode. */
dentry = filter_fid2dentry(cd->obd, NULL, lsc->lsc_gr, lsc->lsc_id);
if (IS_ERR(dentry) || dentry->d_inode == NULL) {
    CWARN("cannot find a dentry for the llog record: "
          LPU64" "LPU64"\n", lsc->lsc_id, lsc->lsc_gr);
} else {
    struct ost_filterdata *ofd;
    LOCK_INODE_MUTEX(dentry->d_inode);
    ofd = dentry->d_inode->i_filterdata;
    /* As LLOG record exists, ofd must be allocated. Epoch might
     * have been changed, but cannot be less than the given one. */
    LASSERT(ofd && ofd->ofd_epoch >= lsc->lsc_ioepoch);
    if (ofd->ofd_epoch == lsc->lsc_ioepoch) {
        OBD_FREE(ofd, sizeof(*ofd));
        dentry->d_inode->i_filterdata = NULL;
        iput(dentry->d_inode);
    }
    UNLOCK_INODE_MUTEX(dentry->d_inode);
}
if (dentry)
    f_dput(dentry);
RETURN(LLOG_PROC_BREAK);
```

9. static int filter_destroy_internal(struct obd_device *obd, obd_id objid, obd_gr group, struct dentry *dparent, struct dentry *dchild)

   Cancel existing llog record if it exists and also unpin the inode if so.

```
LOCK_INODE_MUTEX(inode);
if (inode->i_filterdata) {
    struct ost_filterdata *ofd =
        (struct ost_filterdata *)inode->i_filterdata;
    struct llog_ctxt *ctxt;
    ctxt = llog_get_context(obd, LLOG_SIZE_ORIG_CTXT);
    if (ctxt == NULL) {
        CWARN("llog subsys not setup for SIZE\n");
        RETURN(-ENOENT);
    }
    rc = llog_cat_cancel_records(ctxt->loc_handle, 1,
                                 &ofd->ofd_cookie);
    if (rc) {
        CWARN("Failed to cancel cookie for epoch=%u ino=%lu",
              ofd->ofd_epoch, inode->i_ino);
    }
    OBD_FREE(ofd, sizeof(*ofd));
    inode->i_filterdata = NULL;
```

```
        iput(inode);
    }
    UNLOCK_INODE_MUTEX(inode);
```

6.7 MDC.

1. void mdc_cookies_pack(struct ptlrpc_request *req, int offset, struct md_op_data *op_data)

   Take cookies from @op_data if MF_SOM_CHANGE flag is set.

   ```
   if (op_data->op_flags & MF_SOM_CHANGE) {
       int len = sizeof(struct llog_cookie) * op_data->op_numcookies;
       struct llog_cookie *cookies = lustre_msg_buf(request->rq_reqmsg,
                                                    offset, len);
       memcpy(cookies, op_data->op_cookies, len);
   }
   ```

2. void mdc_setattr_pack(struct ptlrpc_request *req, int offset, struct md_op_data *op_data, void *ea, int ealen)

   Call mdc_cookies_pack() to pack cookies to the request. Cookies and their amount is taken from @op_data instead of explicit attributes.

   ```
   mdc_cookies_pack(req, offset + 4, op_data);
   ```

3. void mdc_close_pack(struct ptlrpc_request *req, int offset, struct md_op_data *op_data)

   Call mdc_cookies_pack() to pack cookies to the request. Adjust capabilities position.

   ```
   mdc_pack_capa(req, offset + 3, op_data->op_capa1);
   mdc_cookies_pack(req, offset + 2, op_data);
   ```

4. int mdc_setattr(struct obd_export *exp, struct md_op_data *op_data, void *ea, int ealen, struct ptlrpc_request **request)

   Get cookies and their amount from @op_data, instead of explicit attributes.

   ```
   int len = sizeof(struct llog_cookie) * op_data->op_numcookies;
   int size[6] = { ..., ealen, len };
   ...
   mdc_setattr_pack(req, REQ_REC_OFF, op_data, ea, ealen);
   ```

5. int mdc_close(struct obd_export *exp, struct md_op_data *op_data, struct obd_client_handle *och, struct ptlrpc_request **request)

   New buffer for llog cookies is added, it is placed before capabilities to keep SOM related buffers close to each other and handle them easier. It is permitted as capabilities are not in production yet.

```
int len = sizeof(struct llog_cookie) * op_data->op_numcookies;
int reqsize[5] = { ..., len, 0 };
...
reqsize[REQ_REC_OFF + 3] = op_data->op_capa1 ?
    sizeof(struct lustre_capa) : 0;
req = ptlrpc_prep_req(class_exp2cliimp(exp), LUSTRE_MDS_VERSION,
```

6. int mdc_done_writing(struct obd_export *exp, struct md_op_data *op_data, struct obd_client_handle *och)

   New buffer for llog cookies is added, it is placed before capabilities to keep SOM related buffers close to each other and handle them easier. It is permitted as capabilities are not in production yet.

```
int len = sizeof(struct llog_cookie) * op_data->op_numcookies;
int reqsize[5] = { ..., len, 0 };
...
reqsize[REQ_REC_OFF + 3] = op_data->op_capa1 ?
    sizeof(struct lustre_capa) : 0;
req = ptlrpc_prep_req(class_exp2cliimp(exp), LUSTRE_MDS_VERSION,
                      MDS_CLOSE, 5, reqsize, NULL);
```

6.8 MDT.

1. static int mdt_setattr_unpack(struct mdt_thread_info *info)

2. int mdt_close_unpack(struct mdt_thread_info *info)

   Unpack obtained llog cookies to mdt_thread_info if the corresponding buffer exists.

```
if (req_capsule_field_present(pill, &RMF_LOGCOOKIES, RCL_CLIENT) &&
    info->mti_epoch->flags & MF_SOM_CHANGE) {
    info->mti_cookie = req_capsule_client_get(pill, &RMF_LOGCOOKIES);
    info->mti_cookie_size =
        req_capsule_get_size(pill, &RMF_LOGCOOKIES, RCL_CLIENT);
}
```

3. static int mdt_txn_stop_cb(const struct lu_env *env, struct thandle *txn, void *cookie)

   Allocate a copy of lov_mds_md and llog cookies for the later llog cookie handling on commit.

```
if (mti->mti_cookie_size) {
    struct md_attr *ma = &mti->mti_attr;
    int size = sizeof(struct llog_cookie) *
        ma->ma_lmm->lmm_stripe_count;
    if (size < mti->mti_cookie_size) {
        CERROR("Too many cookies %d has come, must be <= %d.",
            info->mti_cookie_size / sizeof(struct llog_cookie),
            ma->ma_lmm->lmm_stripe_count);
        return -EINVAL;
    }
    /* lmm & cookies are to be deallocated in commit_cb */
    OBD_ALLOC(txi->txi_lmm, ma->ma_lmm_size);
    if (txi->txi_lmm == NULL)
        return -ENOMEM;
    txi->txi_lmm_size = ma->ma_lmm_size;
    memcpy(txi->txi_lmm, ma->ma_lmm, ma->ma_lmm_size);
    OBD_ALLOC(txi->txi_cookies, size);
    if (txi->txi_cookies == NULL) {
        OBD_FREE(txi->txi_lmm, ma->ma_lmm_size);
        return -ENOMEM;
    }
    memcpy(txi->txi_cookies, mti->mti_cookie,
           sizeof(struct llog_cookie) * mti->mti_cookie_size);
}
```

4. static int mdt_txn_commit_cb(const struct lu_env *env, struct thandle *txn, void *cookie)

   Cancel llog cookies saved in mdt_txn_info and free allocated for the cookies and lov_mds_md memory.

```
size = sizeof(struct llog_cookie) * txi->txi_lmm->lmm_stripe_count;
rc = mdt->mdt_child->md_ops->mdo_llog_cancel(env, mdt->mdt_child,
                                             txi->txi_lmm,
                                             txi->txi_lmm_size,
                                             txi->txi_cookies);
OBD_FREE(txi->txi_cookies, size);
OBD_FREE(txi->txi_lmm, txi->txi_lmm_size);
```

6.9 CMM.

1. static int cmm_llog_cancel(const struct lu_env *env, struct md_device *md, struct lov_mds_md *lmm, int ma_lmm_size, struct llog_cookie *cookies)

   Call mdd method to cancel llog cookies.

   ```
   struct cmm_device *cmm_dev = md2cmm_dev(md);
   return cmm_child_ops(cmm_dev)->mdo_llog_cancel(
       env, cmm_dev->cmm_child, lmm, lmm_size, cookies);
   ```

6.10 MDD.

1. static int mdd_llog_cancel(const struct lu_env *env, struct md_device *md, struct lov_mds_md *lmm, int lmm_size, struct llog_cookie *cookies)

   Call mds method to cancel llog cookies.

   ```
   struct mdd_device *mdd = lu2mdd_dev(&md->md_lu_dev);
   return mds_log_op_cancel(mdd2obd_dev(mdd), lmm, lmm_size, cookies);
   ```

6.11 MDS.

1. int mds_log_op_cancel(struct obd_device *obd, struct lov_mds_md *lmm, int lmm_size, struct llog_cookie *logcookies)

   Prepare lov_stripe_info by the given @lmm and cancel given llog @cookies for it.

   ```
   struct mds_obd *mds = &obd->u.mds;
   struct lov_stripe_md *lsm;
   struct llog_ctxt *ctxt;
   int rc = 0;
   rc = obd_unpackmd(mds->mds_osc_exp, &lsm, lmm, lmm_size);
   if (rc < 0)
       return rc;
   rc = obd_checkmd(mds->mds_osc_exp, obd->obd_self_export, lsm);
   if (rc) {
       obd_free_memmd(mds->mds_osc_exp, &lsm);
       return rc;
   }
   ctxt = llog_get_context(obd, LLOG_MDS_OST_ORIG_CTXT);
   rc = llog_cancel(ctxt, lsm, lsm->lsm_stripe_count, logcookies, 0);
   obd_free_memmd(mds->mds_osc_exp, &lsm);
   return rc;
   ```

6.12 LLOG.

1. void lustre_swab_llog_rec(struct llog_rec_hdr *rec, struct llog_rec_tail *tail)

   Swab new fields properly.

   ```
   __swab64s(&lsc->lsc_id);
   __swab64s(&lsc->lsc_gr);
   ```

6.13 LibLustre.

1. int llu_local_open(struct llu_inode_info *lli, struct lookup_intent *it)
2. void llu_epoch_open(struct llu_inode_info *lli, int ioepoch)
3. int llu_sizeonmds_update(struct inode *inode, struct lustre_handle *fh, __u64 ioepoch)
4. int llu_md_close(struct obd_export *md_exp, struct inode *inode)
5. static void llu_ap_fill_obdo(void *data, int cmd, struct obdo *oa)
6. int llu_inode_getattr(struct inode *inode, struct obdo *obdo)
7. int llu_md_setattr(struct inode *inode, struct md_op_data *op_data)
8. int llu_setattr_raw(struct inode *inode, struct iattr *attr)
9. static int llu_setattr_done_writing(struct inode *inode, struct md_op_data *op_data)
10. static void llu_truncate(struct inode *inode, obd_flag flags)

These methods are changed/added similar to the LLITE code.

7 Protocol, APIs, disk format.

1. mdt_close_client format changes.

   New LOGCOOKIES buffer is added before CAPA buffer. It is allowed to add not to the end as capabilities are not in production yet.

   ```
   static const struct req_msg_field *mdt_close_client[] = {
       &RMF_PTLRPC_BODY,
       &RMF_MDT_EPOCH,
       &RMF_REC_SETATTR,
       &RMF_LOGCOOKIES,
       &RMF_CAPA1
   };
   ```

2. MD API changes.

   m_setattr() does not have separate ea2 & ea2len anymore,cookies are passed through md_op_data instead.

3. OBD API changes.

   o_prep_async_page() gets the stripe number instead of the exact lov_oinfo pointer. It allows it to get not lov_oinfo data from lov_stripe_info, but the appropriate log_cookie too.

4. MD Device API changes.

   mdo_llog_cancel() is added to pass llog cookies to be canceled through MD Device stack from MDT to MDS through CMM and MDD.

8 Scalability and performance.

The current DLD is a part of Size-on-MDS feature which is about llog'ing object attribute change on OSTs to guarantee these attributes are not lost somewhere on the way to MDS. It has neither scalability or performance implications by itself.

The Size-on-MDS feature itself allows clients to avoid making RPCs to the OSTs to get the attributes encoded in the file objects what results in the significantly improved performance of listing directories.

9 Test plan.

Test plan is out of scope of this document and will be elaborated as a separate task.

10 Recovery.

Test plan is out of scope of this document and will be elaborated as a separate task.

11 Focus for inspection.

11.1 OST LLOG Cancel.

1. LLOG processing for every cancel is too slow, it read all the llog since the beginning through the wanted record. It must be optimized by introducing some cookie->inode pointer hash.

2. LLOG Cancels seems to be handled by LDLM (ldlm_cancel_handle()) instead of OST (ost_handle()), but we need not only to cancel LLOG records, but unpin inodes on OST too.

11.2 MDT Stack and LLOG Interface.

There is a need to pass some attributes to mdd_txn_stop_cb() and create lov_stripe_info there, later on mdd_txn_commit_cb() use this info to send out llog cancels to corresponding OSTs.

11.3 OST LLOG.

Ensure LLOG is used properly, every attribute change for a distinct epoch of every file is lloged and canceled properly.

12 Alternatives.

12.1 ll_sizeonmds_update() optimization.

There is no need to send OBD_GETATTR RPC to all the OST, but only to those whose [0;EOF] locks are absent. It can be achieved by getting llog cookies on [0;EOF] lock enqueue, and put them to lsm. the further request for the SOM attribute update, obtain only cookies from OST with no [0;EOF] locks, other are taken from lsm. Even if new epoch has been already started, and the cookies kept in lsm are of the current epoch, MDS skip them if the SOM attribute update itself is for the previous epoch. On the other hand, if SOM attribute update is for the last closed epoch, and therefore llog cookies are correct as well, MDS will send cancels for all the received cookies.