Grants

Jan 22, 2007

1 Introduction

This document mainly explains how space grants currently work based on v1_5_97 (Refer bug 2800 and bug 974 for more details) and also propose several alternative strategies for support grants under I/O scheduler.

2 Requirements

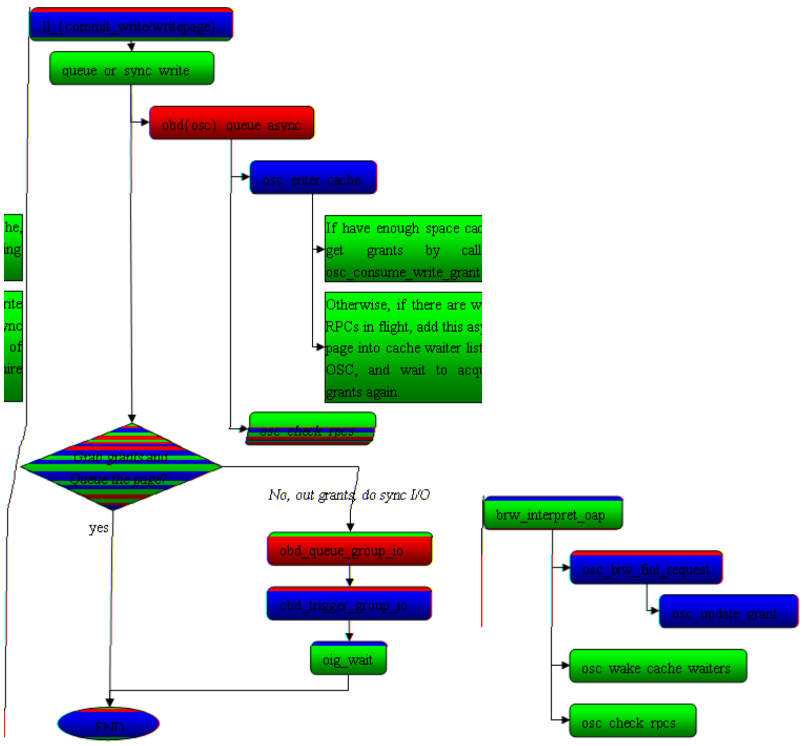
Handle ENOSPC in case of introduction of client-side writeback cache especially when the OSTs' disk is extremely full.

3 Functional specification

- When filesystem is nearly full and clients are writing hard, a I/O request should return ENOSPC.
- If a client times out on OST and reconnects it doesn't have its local grant revoked.
- Client should get an initial grant to avoid initial sync I/O.
- Shrink grants on the inactive clients.
- Intellect space grants management.

4 Use cases

4.1 Normal case (v1_5_97)


ll_(commit_write/writepage)
queue or sync write
obd(osc) queue async
osc enter cache
If have enough space cac get grants by call osc_consume_write_grant
Otherwise, if there are w RPCs in flight, add this as page into cache waiter list OSC, and wait to acqu grants again.
osc check rpcs
Grab grants and Queue the page?
No, out grants, do sync I/O
yes
obd_queue_group_io
obd_trigger_group_io
oig_wait
END
brw_interpret_oap
osc_brw_fini_request
osc update grant
osc wake cache waiters
osc check rpcs

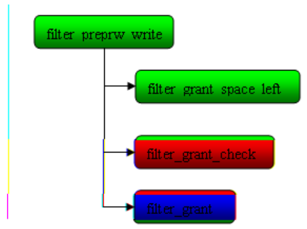
- In *ll_commit_write or ll_writepage* (for mmap write), call queue_or_sync_write to grab space grants;
- If the cached, unwritten client data doesn't reach the limitation *cl_dirty_max* and there are enough space grants, consume the available grants of the OSC and mark the asynchronous page as OBD_BRW_FROM_GRANTED (in *osc_enter_cache*);
- Otherwise, if there are some write RPCs in flight, add the asynchronous page to the cache waiter list of the OSC which ran out available grants. Wait until waking up by I/O completion (in *brw_interpret_oap*);
- If grab grants, queue the asynchronous page and build a optimal I/O RPC if possible by calling *osc_check_rpcs*.
- If still out of grants, do synchronous I/O by *obd_queue_group_io*/*obd_trigger_group_io*/*oig_wait*, which would drain away all queued dirty pages and wait until I/O completion of the page out of grants.

- Before putting I/O RPC into wire, call *osc_announce_cached* to compute the how many grants should be acquired from OST. The grants amount is equal to Max.(cli_dirty_max, max_rpc_in_flight * max_size_per_rpc).
- Upon I/O completion (in *brw_interpret_oap*) , it will first update the available grants of OSC, and then call *osc_wake_cache_waiters* to wake up cache waiters. In *osc_wake_cache_waiters*, it will grant space to the waiting asynchronous pages; if the updated grants is not enough as there are lots of asynchronous pages waiting for grants and moreover there are no pending RPC in flight to return grants, it will return -*EDQUOT* and wake up the cache waiter, let it do sync IO.

4.2 Startup/Recovery (v1_5_97)

- During connection or reconnection for recovery, it will mark the connection flags with OBD_CONNECT_GRANT. For initial connection, it will request grants amount of 2 * *cl_max_pages_per_rpc* << *PAGE_SHIFT* (*2M*); For reconnection of recovery grants, it will request grants amount of *cl_avail_grant* before disconnection.
- When OST receive connection request with grants intention, it will return certain space grants according to the left space.

4.3 Disk extremely full (v1_5_97)



- After OST receives an I/O request, it will first figure out the left space exclude the space granted out to clients and compensate for ext3 indirect block overhead (in *filter_grant_space_left*).
- When clients uses out space grants from OSTs they fall through to sync writes. The pages in the sync writes haven't been acquired grants (not marked as OBD_BRW_FROM_GRANTED) and will error with ENOSPC if there isn't room in the filesystem for them after grants are taken into account according to the available space returned in the previous step. However, the writeback pages that are usually already acquired space grants can write right on through. (in *filter_grant_check*)
- If the object is marked as OBD_MD_FLGRANT, it will call *filter_grant* to calculate how much space grant to allocate to the client based on the left space. The grants amount is:

```
want: how much space grants client wants to acquire.
fs_space_left: available space.
grant = min((want >> blockbits), (fs_space_left >> blockbits) / 8);
grant <<= blockbits;
if (grant) {
    if ((grant > 2M) && (!obd->obd_recovering))
         grant = 2M;
     ...
}
```

5 Logic specification

N/A

6 State management

6.1 Scalability & performance

In the v1_5_97, the client is usually not ran out of grant unless the filesystem is full or there are many small writes from various files use out of grants but not trigger any IO. The reason is that: On the initial connection, it will be granted amount (2M) and it will start I/O as soon as optimal RPC can be built as pages are dirtied; And the grant amount will be refreshed to be more than the dirty data sent upon the I/O completion.

6.2 Locking changes

As grants are a limited resource in real installations, so it's no reasonable to give out lots of space grants (GBs) to client. But we can use grant lock to make OST more intellect to manage the space grants:

- During initial connection or reconnect from disconnect recovery, OSC acquires space grant lock from OST. If available space is still large, OST will return lots of space grants to OSC which is almost fill client's physical memory.
- Upon receiving I/O requests, OST will return certain reasonable space (*cli_phy_mem* - *cli_avail_grants*) to client to avoid out of grants.
- When disk of OST is extremely full and lower than certain threshold, OST shrinks the granted space of some clients by lock callback. In this case, OST just gives out small space grants.
- When the left space grows up own to object deletions and exceeds certain threshold, OST will recovery to give out large space grants.

7 Alternatives

In the I/O scheduler, it appears a very bad performance after add the support for grants/quota. The main reason is that: To get client closer to regular linux file system the new I/O path is VM driven, it doesn't start any IOs until use out of client's available grants. Thus, when out of grants/ space cache, there are usually lots of dirty pages in I/O queues. At this time, if wait until the completion of draining away all dirty pages, it would take ages.

To fix this bugs, there are two proposal strategies:

7.1 Wait until acquire grants or the page writeback

- When out of grants in *ll_commit_write*, we don't always wait until the completion of last asynchronous page which is triggered out of grants. Instead, we just wait until either acquire grants again or the page writeback.
- When out of grants, queue the asynchronous page with flags ASYNC_URGENT, and trigger the synchronous I/O start to push the pages in dirty list under writeback immediately. After that, Add this asynchronous page into cache waiter list of corresponding OSC.
- If the page has already been under writeback, wake up the cache waiter and let it waits until the page writeback by kernel function *wait_on_page_writeback*.
- If some I/O completion updates the available grants of OSC, and grant space to the asynchronous page, clear the flags ASYNC_URGENT, wake up the cache waiter and queue it to dirty list, let VM determine what time to write out it.

Compared with original I/O scheduler with grants support, the advantage is:

- Usually wait only one I/O RPC completion when out of grants.

The disadvantage is:

- Even there is lots of space left on OSTs it also will use out of grants and start synchronous I/O. And it doesn't make better use of the VM system in linux kernel.

7.2 Integrate with extent lock BST

This strategy is based on green's patch on Bug 10718: slow lock cancellation due to excessive page walking. In the patch, all dirty pages also manage by the extent lock, so when extent lock revocations it needn't walk through all dirty pages. The strategy is as following:

- During OSC installation, start a kernel daemon which uses to write out data of extent locks as reaching the limitation of dirty data per OSC.
- In *ll_commit_write*, If out of grants, wake up the daemon to start I/O. Then add the asynchronous page out of grants to the cache waiter list of the OSC. Wait until acquire grants again or the page writeback.
- If grab grants, queue this I/O and check whether the dirty data of the OSC reaching 3 * max_dirty_data /4. If so, wake up the daemon to start I/O on the OSC by the way similar with the extent lock callback.

8 Focus for inspections