LNET SMP performance improvement

Liang Zhen

5 August 2008

1 Introduction

As we know, LNet has only one global lock: LNET_LOCK, which is spinlock in kernel space. All descriptors and states are protected by this lock, most time LNet is just single-thread accessible. It's not a problem if LNet operations are light, however we can't always guarantee this, for example, client RDMA with a lot of servers at same time, and some of these peers are running in heavy load, there could be a lot of MEs queued for these busy peers on client, some incoming messages have to loop for hundreds of times to find matching MD, any access to LNet will have to busy wait on LNET_LOCK() for long time. Also, handle operations and peer operations can be time consuming on scaled system too.

2 Architecture

There are two ways to improve scalability of LNet:

- Matching MD be scalable
- Grained locking for LNet

2.1 Matching MD be scalable

Searching on hash will be much faster then searching on list, however, a lot of information involved in MD matching: source NID and PID, match_bits, ignore_bits, not all of them are accurate matching (i.e, match ID can be LNET_NID_ANY), so we must be very careful about hash algorithm, otherwise we could get unexpected dropping.

Also, LNet inherited some features from Portals: it's legal to queue request buffer(wildcast match_id, without match_bits) and RDMA buffer(with specific match_id and match_bits) on the same portal matching list (ptl_ml), it can support inserting of ME(LNetMEInsert, LNetMEAttach(...LNET_INS_BEFORE)). It's impossible to support these features with

hash table. Fortunately, in all Lustre cases, we never use these features, different MD always on different portals, and we never insert any ME.

So it's reasonable for us to abandon these features on RDMA portal, of course, we can still support them on request portal.

Here are more details about hash algorithm:

- Request portal: still queue ME on ptl_ml, because match_id is always LNET_NID_ANY, also, user still can LNetMEInsert, LNetMEAttach(... LNET_INS_BEFORE) ME on request portal
- RDMA (Reply and bulk) portal: ME will be queued on hash table(ptl_mhash), entries of hash are indexed by (match_id.nid % LNET_PORTAL_HASH_SIZE)
- It's invalid to queue RDMA buffer on request portal, it's invalid to queue request buffer on RDMA portal.
- Matching always loop on ptl_ml if it's a request portal, and on ptl_mhash if it's a RDMA portal
- Type of portal (RDMA or request) can be decided by the first attached ME

2.2 Grained locking

If we can grain the global lock to more locks(without increase too many additional lock/unlock operations), LNet can be more scalable because not all operations need to race on the same global lock (LNET_LOCK).

Current LNET_LOCK protects:

1. handle operations of all LNet descriptors (EQ, ME, MD)
2. operations of all LNet descriptors (Attach, Bind, Unlink, match on portals)
3. peer table
4. credits and LNet message
5. router buffer
6. stats counter of LNet

2.2.1 peer table & credits & router buffer

Operations on these data are very tight, also, these operations have few overlap with other operations(operations on LNet descriptors like MD, ME), so we can have one lock to protect them and still call it lnet_lock. For example, we hold lnet_lock(), find peer and get credit, then lnet_unlock(), we will see more detail later.

2.2.2 stats counter

Stats counter could be called in any context, these operations are very fast, so we create another lock to protect global data like counters, let's name it as lnet_atomic_lock. It's legal to race lnet_atomic_lock with hold of any other locks.

2.2.3 handle operations & LNet descriptors operations

They are the most time consuming operations in LNet, it would help for scalability if we can divide these operations to different logic parts.

Although these operations are very tight, but as we know there are two types of MDs in LNet: MD on portal(created by LNetMDAttach), MD not on portals(created by LNetMDBind). Matching MD operation always happen on MD on portal, so it's possible for us to have individual lock for matching operation. Let's assume we have two locks here:

- lnet_portal_lock : protect ME and MD on portal
- lnet_obj_lock : protect MD (not on portal), EQ etc.

Also, we will have two handle hash tables:

- EQ and MD share one handle hash table, which is protected by lnet_obj_lock
- ME have individual handle hash table, which is protected by lnet_portal_lock

It's legal to hold lnet_obj_lock and race lnet_portal_lock, so:

- All EQ operations need hold lnet_obj_lock
- All MD operations need hold lnet_obj_lock
- All ME operations need hold lnet_portal_lock
- LNetMDAttach needs hold both lnet_obj_lock & lnet_portal_lock
- LNetMDUnlink needs hold both lnet_obj_lock & lnet_portal_lock if MD is on portal, otherwise only needs lnet_obj_lock
- LNetMEUnlink needs hold both lnet_obj_lock & lnet_portal_lock
- MD NOT on portal, lnet_commit_md needs hold lnet_obj_lock
- MD on portal, lnet_commit_md needs hold lnet_portal_lock
- MD NOT on portal, lnet_finalize needs hold lnet_obj_lock

- MD on portal, lnet_finalize needs hold both lnet_obj_lock & lnet_portal_lock
- lnet_try_match_md only needs hold lnet_portal_lock (md_unlink can be delayed to lnet_finalize)

With these assumptions, we can see that:

- lnet_parse_get() & lnet_parse_put() will not race with lnet_parse_ack() & lnet_parse_reply() on lnet_obj_lock
- lnet_parse_get() & lnet_parse_put() will not race with LNetPut() & LNetGet() on lnet_obj_lock.
- LNetMDBind & LNetMDUnlink(if MD is not on portal) will not race with ME operations

So we would benefit from these changes.

2.2.4 Conclusion

We will have four locks:

- lnet_lock
- lnet_obj_lock
- lnet_portal_lock
- lnet_atomic_lock

Also, we have additional benefit from this change, because handle operations can be faster because we have more hash tables for handle(ME has individual hash).

3 External Functional specifications

N/A (There is no change for external access)

4 High Level Logic

4.1 Matching hash

We will change lnet_portal_t and lnet_me_t to :

```
#define LNET_PTL_COMPAT
typedef struct {
    struct list_head *ptl_mhash;
    struct list_head  ptl_ml;
    ......
    unsigned int ptl_options;
} lnet_portal_t;
```

We still keep ptl_ml for request portal

4.2 Locks

We have elaborated how to grain locks in section 2.2, now we provide more details by some examples:

4.2.1 Outgoing message (LNetPut)

```
lnet_obj_lock()  //  LNET_LOCK
md = handle2md()
commit_md()
...
lnet_obj_unlock()
lnet_stat_lock()
counters operations
lnet_stat_unlock()  // LNET_UNLOCK
......
lnet_lock()         // LNET_LOCK
find peer of target or route
get credit
lnet_unlock()       // LNET_UNLOCK
ni_send
```

4.2.2 Incoming message (LNET_MSG_PUT for lnet_parse)

```
lnet_lock()     // LNET_LOCK()
find rx peer
if (forward) {
        get router buffer and credit
        lnet_unlock()  // LNET_UNLOCK()
        ni_send()
        return
}
lnet_unlock()    // LNET_UNLOCK()
...
```

```
lnet_portal_lock()      // LNET_LOCK
match_md
lnet_portal_unlock()    // LNET_UNLOCK
ni_recv()
```

4.2.3 lnet_finalize()

```
lnet_obj_lock()        // LNET_LOCK
if ((md->md_options &  (LNET_MD_OP_GET | LNET_MD_OP_PUT)) != 0) // MD on portal
    lnet_portal_lock()
md->md_refcount--;
unlink = lnet_md_unlinkable(md);
if ((md->md_options &  (LNET_MD_OP_GET | LNET_MD_OP_PUT)) != 0) // MD on portal
    lnet_portal_unlock()
unlink_md
enq_event
lnet_obj_unlock()
.....
lnet_lock()
if (ack) {
       return credits
       lnet_unlock()   // LNET_UNLOCK
       ...
       lnet_send()
       lnet_lock()     // LNET_LOCK
} else if (forward) {
       lnet_unlock()   // LNET_LOCK
       lnet_send()
       lnet_lock()     // LNET_UNLOCK
}
return credits
lnet_unlock()          // LNET_UNLOCK
```

4.2.4 LNetMDBind()

```
lnet_obj_lock()   // LNET_LOCK
......
lnet_obj_unlock() // LNET_UNLOCK
```

4.2.5 LNetMDAttach

```
lnet_obj_lock()    // LNET_LOCK
lnet_portal_lock()
......
lnet_match_blocked_msg(md, matches, drops)
```

```
lnet_portal_unlock()
lnet_obj_unlock()  // LNET_UNLOCK
```

4.2.6 lnet_try_match_md()

```
......
if ((md->md_flags & LNET_MD_FLAG_AUTO_UNLINK) != 0 &&
    lnet_md_exhausted(md)) {  // detach MD from portal
        md->md_me = NULL;
        me->me_md = NULL;
        /* ME gets unlinked if required */
        if (me->me_unlink == LNET_UNLINK) {
            list_del(&me->me_list);
            lnet_invalidate_handle(&me->me_lh);
            lnet_me_free(me);
        }
}
return LNET_MATCHMD_OK;
```

5 Use-Case Scenarios

Although we add few limitation for LNet users, but there is actually no change for current use-cases, because Lustre never uses these features.

- Request buffer and RDMA buffer can't be attached on the same portal, LNetAttachME() will return -EPERM if upper layer trys to attach RDMA buffer on request portal or attach request buffer on RDMA portal
- LNetMEInsert() and LNetMEAttach(... LNET_INS_BEFORE...) are only permitted on request portal, otherwise -EPERM will be returned

6 Test Plan

(more detail from Nikita)

7 Review

- Is the hash algorithm reasonable
- Is it harmless if we delay lnet_md_unlink()(mark the MD as zombie and unlink from handle hash table) from lnet_try_match_md() to lnet_finalize()?