HLD For SMP node affinity

Introduction

Current versions of Lustre rely on a single active metadata server. Metadata throughput may be a bottleneck for large sites with many thousands of nodes. System architects typically resolve metadata throughput performance issues by deploying the MDS on faster hardware. However, this approach is not a panacea. Today, faster hardware means more cores instead of faster cores, and Lustre has a number SMP scalability issues while running on servers with many CPUs. Scaling across multiple CPU sockets is a real problem: experience has shown in some cases performance regresses when the number of CPU cores is high.

Over the last few years a number of ideas have been developed to enable Lustre to effectively exploit multiple CPUs. The purpose of this project is to take these ideas past the prototype stage and into production. These features will be benchmarked to quantify the scalability improvement of Lustre on a single metadata server. After review, the code will be committed to the Master branch of Lustre.

Definitions

Solution Architecture For SMP node affinity contains definitions that the reader may find provide helpful background.

CPU partition and CPU partition table

A CPU partition (CPT) is conceptually similar to the 'cpuset' of Linux. However, in Project Apus, the CPU partition is designed to provide a portable abstract layer that can be conveniently used by kernel threads, unlike the 'cpuset' that is only accessible from userspace. To achieve this a new CPU partition library will be implemented in libcfs.

CPT table is a set of CPTs:

- A CPT table can contain 1-N CPU partitions
- CPUs in one CPT shouldn't overlap with CPUs in CPTs of the same CPT table
- A CPT table can cover all CPUs in the system, or a subset of system CPUs.

NUMA allocators

Linux kernel has interfaces for NUMA allocators (`kmalloc_node`). MDS may be configured with a partition that contains more than one node. To support this case, Project Apus NUMA allocator will be able to spread allocations over different nodes in one partition.

LNet Event Queue

LNet is an async message library. Message completion events are delivered to upper layer via Event Queue (EQ) either by callback or by Polling event. Lustre kernel modules reply on EQ callbacks of LNet to get completion event.

LNet wildcard buffer and unique buffer

There are two types of buffers for passive messages:

- **Wildcard buffer** sizable to any incoming data on that LNet Portal (service ID) if size can match.
- **Unique buffer** has unique identifier (match-bits and source NID) and can only be used to as sink buffer for message with exact same match-bits from specified peer (NID).

Ptlrpc service

Server side of Lustre service, each service contains these key components:

- Service threads pool
- Service threads waitq
- Request buffer pool
- Request portal

- Reply portal

Changes from Solution Architecture

None.

Functional specification

CPU partition APIs

```
struct cfs_cpt_table *cfs_cpt_table_alloc(int nparts)
```

Allocate an empty CPT-table which with @nparts partitions. Returned CPT table is not ready to use, caller need to setup CPUs in each partition by cfs_cpt_table_set_*().

```
cfs_cpt_table_free(struct cfs_cpt_table *ctab)
```

Release CPT-table @ctab

```
cfs_cpt_set/unset_cpu(struct cfs_cpt_table *ctab, int cpt, int cpu)
```

Add/remove @cpu to/from partition @cpt of CPT-table @ctab

```
int  cfs_cpt_set/unset_cpumask(struct cfs_cpt_table *ctab, int cpt, cpumask_t *mask)
```

Add/remove all CPUs in @mask to/from partition @cpt of CPT-table @ctab

```
void cfs_cpt_set/unset_node(cfs_cpt_table_t *ctab, int cpt, int node)
```

Add/remove all CPUs in NUMA node @node to/from partition @cpt of CPT table @ctab

```
int cfs_cpt_current(struct cfs_cpt_table_t *ctab, int remap)
```

Map current CPU to partition ID of CPT-table @ctab and return. If current CPU is not set in @ctab, it will hash current CPU to partition ID of @ctab if @remap is true, otherwise it just return -1.

```
struct cfs_cpt_table *cfs_cpt_table_create(int nparts)
```

Create a CPT table with total @nparts partitions, and evenly distribute all online CPUs to each partition.

```
struct cfs_cpt_table *cfs_cpt_table_create_pattern(char *pattern)
```

Create a CPT table based on string pattern @pattern, more details can be found in “Use cases”.

Per-cpu-partition-lock (percpt-lock)

Percpt-lock is similar with percpu lock in Linux kernel, it’s a lock-array and each CPT has a private-lock (entry) in the lock-array. Percpt-lock is non-sleeping lock.

While locking a private-lock of percpt-lock (one entry of the lock-array), user is allowed to read & write data belonging to partition of that private-lock, also user can read global data protected by the percpt-lock. User can modify global data only if he locked all private-locks (all entries in the lock-array).

`struct cfs_percpt_lock *cfs_percpt_lock_alloc/free(struct cfs_cpt_table *ctab)`

Create/destroy a percpt-lock.

`cfs_percpt_lock/unlock(struct cfs_percpt_lock *cpt_lock, int index)`

Lock/unlock CPT @index if @index >= 0 and @index < (number of CPTs), otherwise it will exclusively lock/unlock all private-locks in @cpt_lock.

NUMA allocators

`void *cfs_cpt_malloc(struct cfs_cpt_table_t *ctab , int cpt, size_t size, unsigned gfp_flags)`

Allocate memory size of @size in partition @cpt of CPT-table @ctab. If partition @cpt covered more than one node, then it will evenly spread the allocation to any of those nodes.

`void *cfs_cpt_vmalloc(struct cfs_cpt_table_t *ctab, int cpt, size_t size)`

Vmalloc memory size of @size in partition @cpt of CPT table @ctab. Same as above, if partition @cpt covered more than one node, then it will evenly spread the allocation to any of those nodes.

`cfs_page_t *cfs_page_cpt_alloc(struct cfs_cpt_table_t *ctab, int cpt, unsigned gfp_flags)`

Allocate a page in partition @cpt of CPT table @ctab. If partition @cpt covered more than one node, then it will evenly spread the allocation to any of those nodes.

Change CPU affinity of thread

`cfs_cpt_bind(struct cfs_cpt_table_t *ctab, int cpt)`

Change CPU affinity of current thread, migrate it to CPUs in partition @cpt of CPT table @ctab. If @cpt is negative then current thread allow to run on all CPUs in @ctab. If there is only one CPT and it covered all online CPUs, then no affinity is set.

Initializer of ptlrpc service

`Ptlrpc_init_svc(struct cfs_cpt_table *ctab, int *cpts, int ncpt, …)`

We add three new parameters to ptlrpc_init_svc()

- **ctab** is the target CPT table for setting CPU affinity of service. To specify no-affinity service pass in NULL as @ctab.
- **cpts** is an array of partition IDs. If @cpts is NULL, create service threads for all partitions of @ctab. If @cpts is not NULL, only create service threads in those partitions contained by @cpts.
- **ncpt** the length of @cpts array if @cpts is not NULL.

Create CPT-table by String Pattern Example

`cfs_cpt_table_create_pattern(char *pattern)`

For example:

```
char *pattern="0[0-3] 1[4-7] 2[8-11] 3[12-15]"
```

`cfs_cpt_table_create_pattern()` will create a CPT table with four partitions, the first partition contains CPU[0-3], the second contains CPU[4-7], the third contains CPU[8-11] and the last contains CPU[12-15].

Logic specification

CPT table

Libcfs will create a global CPT-table (cfs_cpt_table) during module load phase. This CPT-table will be referred by all Lustre modules.

If user is using string pattern to create CP- table, there is no special code path and the CPT table will be created as required.

If user wants to create CPT table with number of partitions as parameter, then the steps are:

- Set hyper-threads of a same core in the same CPT
- Set cores of a same CPU socket in the same CPT

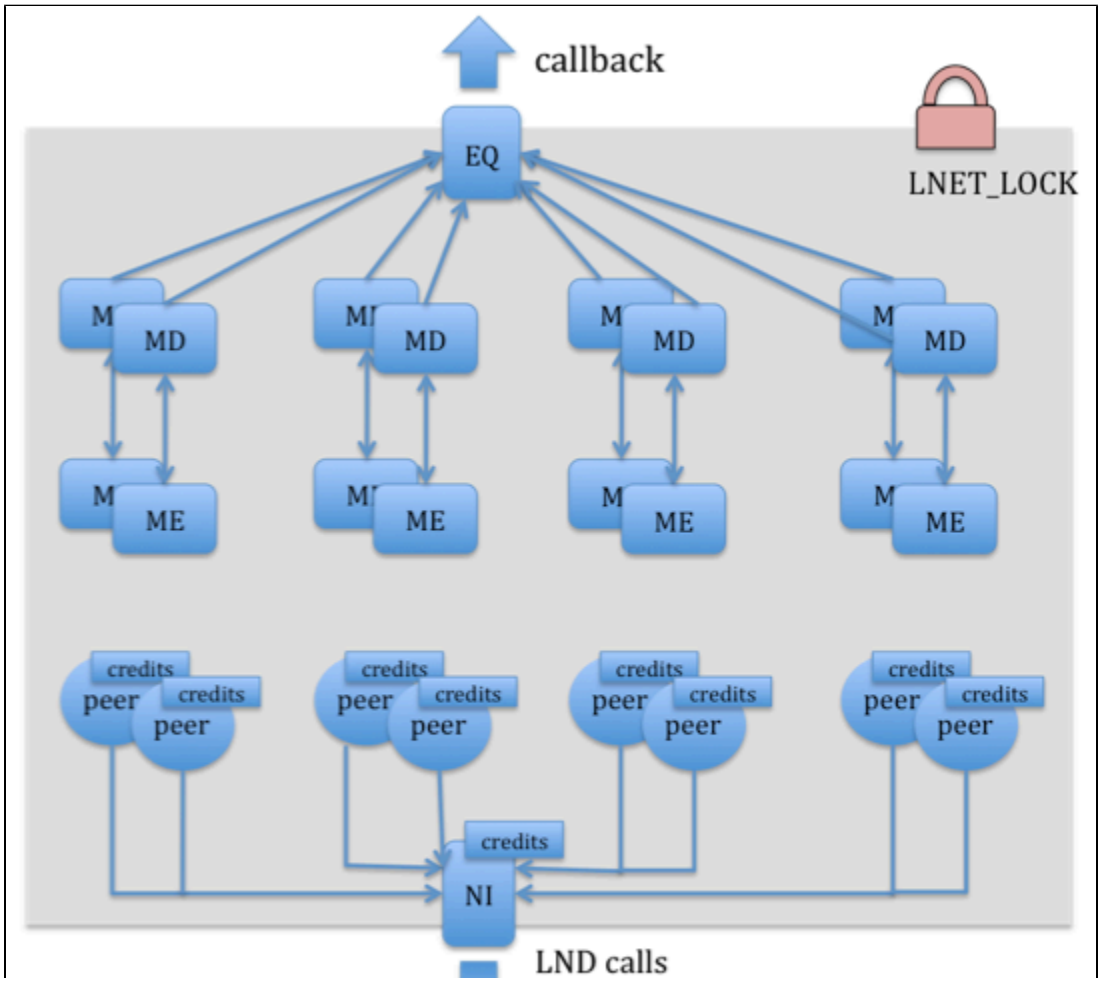
Finer-grain locking for LNet

Current LNet is protected by a global spinlock LNET_LOCK (Graph-1):

- Maintain LNet descriptors (MD, ME, EQ).
- Serialize LNet Event (EQ callbacks are serialized by this lock).
- Global peer table.
- Credits system (NI credits, peer credits).

This global spinlock became to a significant performance bottleneck on fat CPU machine. Specific issues with the spinlock include:

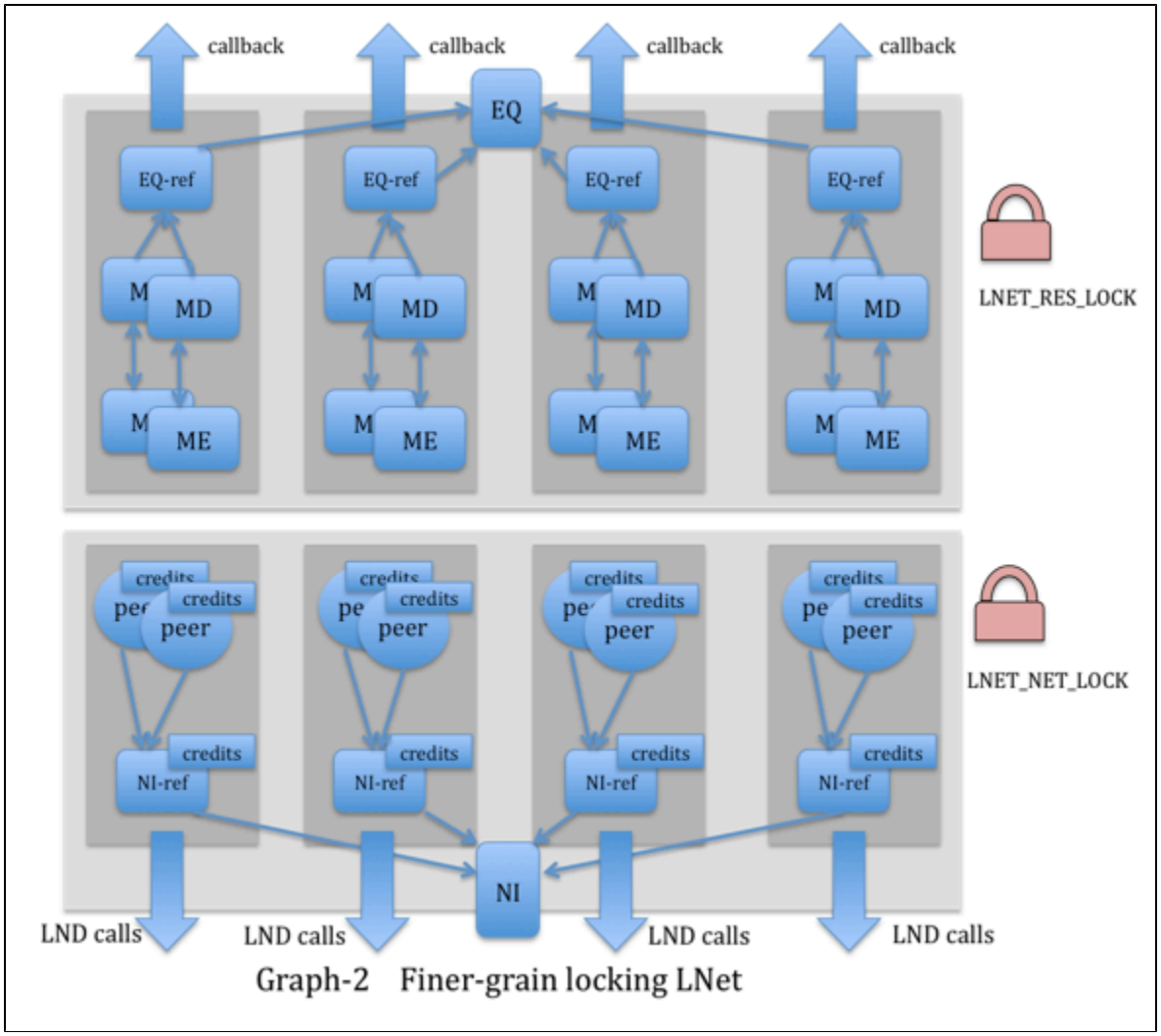
- All LND threads will contend on this lock for credits system and peer table.
- All LND threads and All ptlrpc service threads will contend on this lock while accessing LNet descriptors:
    - LND threads call lnet_parse to match buffers
    - LND threads call lnet_finalize to enqueue event and callback to upper layer.
    - Ptlrpc service threads contend with EQ callbacks on service request queue.
    - Ptlrpc service threads call LNetPut/Get which needs this lock as well.


callback
EQ
LNET_LOCK
MD
ME
credits
peer
NI
LND calls

Graph-1 LNET protected by single-lock

To improve this situation the global LNET_LOCK will be replaced with two percpt-locks (Graph-2):

- **LNET_NET_LOCK(id)** the purpose is to protect peer table and credits system. Peers are hashed to different CPTs by NID. In addition, NI-credits will be replaced by CPT NI-credits:
  - A thread only needs to hold one private-lock of percpt-lock while accessing peer-table and credits system, instead of locking the whole LNet.
  - There are a few operations that need to take exclusive lock, for example: adding or removing a NI. These operations are rare and are not expected to be a performance bottleneck.
- LNET_RES_LOCK(id)
  ME, MD and EQ are protected by LNET_RES_LOCK:
  - ME/MD for unique buffer are hashed to different CPTs by source NID
  - ME/MD for wildcard buffer will directly be attached on current CPT.
  - Create/unlink EQ needs to exclusively lock LNET_RES_LOCK
  - EQ has percpt refcount, which means MD on any CPT can get/put reference of EQ without taking a exclusive lock
  - EQ callback is protected by private-lock
  - Create/unlink EQ needs to exclusively lock LNET_RES_LOCK, which is rare.



Graph-2 Finer-grain locking LNet

LNet Event Queue

As described above, EQ callback will be protected by private lock of percpt-lock instead of a global lock:

- LNet could have parallel EQ callbacks for different buffers (MD).

- Order of EQ callbacks for the same buffer (MD) will still be guaranteed (i.e: callback for UNLINK event is the last one), because one MD can only be attached on one CPT.
- LNetEQPoll() is still ordered by a global spinlock because we need to protect enqueue/dequeue operation on the event queue. This means scalability of LNetEQPoll() will not be improved. Fortunately, Lustre replies on LNetEQPoll rarely.
- If an EQ is created with LNetEQAlloc(0, ...) (EQ count is 0), then event wouldn't be pushed into event-queue and it will only notify upper layer by callback, which means there is no overhead of global lock, also means LNetEQPoll will return nothing. We encourage user to use EQ in this way (callback only) in multiple threads environment.

LNet passive buffer post.

As described in previous sections:

- Unique buffer ME & MD will be attached to different partitions by hashing source NID, LND thread can get source NID from lnet_msg_t and find sink buffer in the partition hashed by NID(lnet_parse).
- Wildcard buffer ME & MD are always attached on partition that calling thread is running on, LND thread will try to match buffer on local partition as well.
    - If LNet can not find buffer on local partition, it will try to "steal" a buffer from other partitions.
    - There is a "active" flag for each partition of LNet, If there is a long period that no thread posted any buffer on a partition, and there is no buffer left on this partition, LND thread will mark it as "inactive" while calling into LNet (lnet_pasre), and search buffer in any of other "active" partitions.
    - If LNet can not find a buffer on any partition, and the target portal is lazy-portal, then the message will be attached to the blocked message queue.
    - Blocked message queue is shared by all partitions and serialized by a global lock.
    - Upper layer threads are expected to post enough buffers (see ptlrpc_check_rqbd_pool() for detail), "blocked message" processing is a rare path and is not expect to become a performance issue.

LND threads pool

By default, number of LND threads equals to number of hyper-threads (HTs) on a system. This number is excessive if there are tens or hundreds of cores with HTs but few network interfaces. Such configuration will potentially hurt performance because of lock contentions and unnecessary context switching.

In this project, the number of LND threads will rely on number of CPTs instead of CPU cores (or HTs). Additional threads will only be created if more than one network interfaces (hardware network interface, not LNet interface) is available.

We will set CPT affinity for LND threads by default (bind threads on CPU partition).

Ptlrpc service

ptlrpc_service will be restructured. Many global resources of service will be changed to CPT resources:

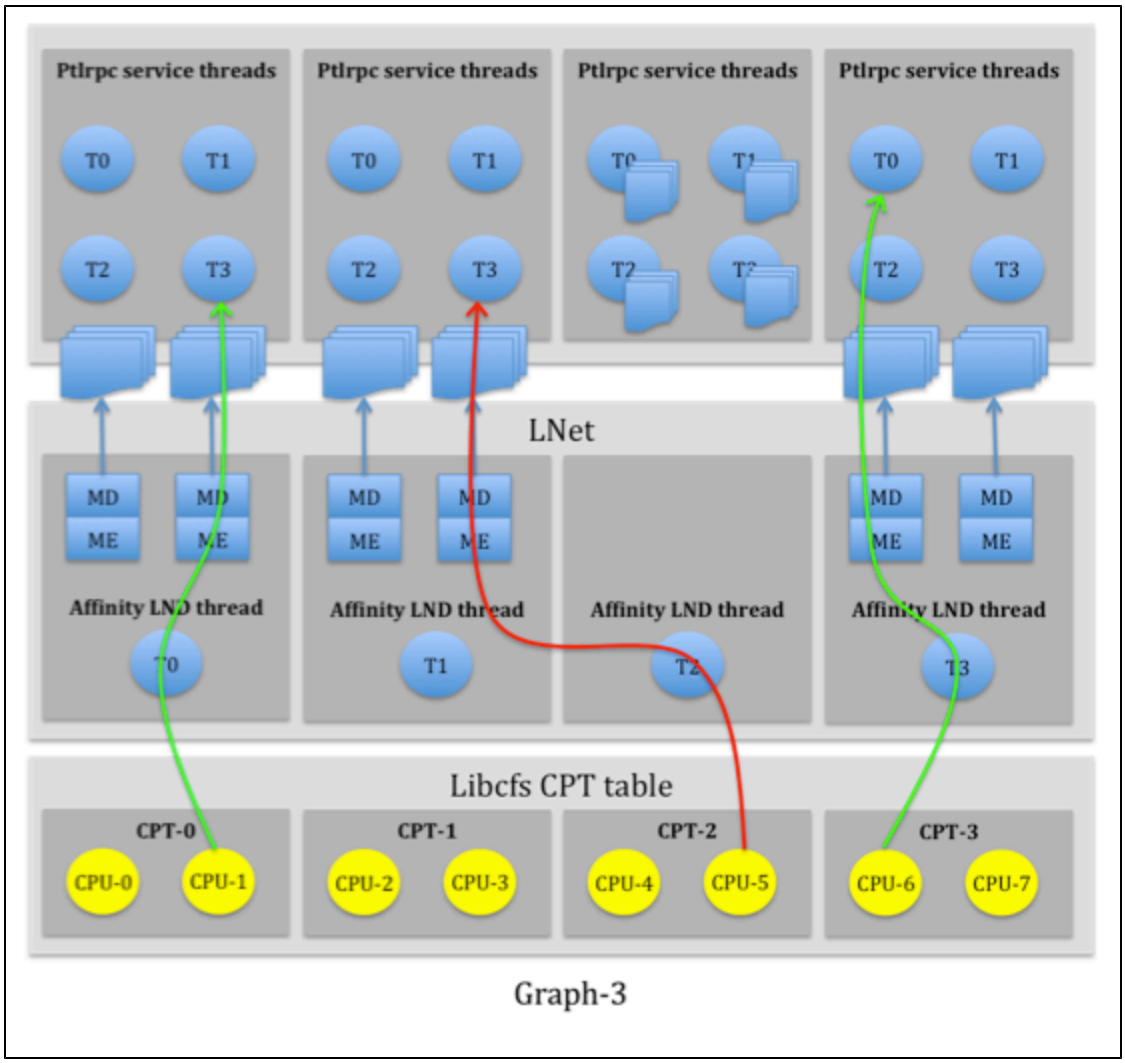
- Request buffer pool
- Incoming request queue
- Service threads pool
- Service threads wait-queue
- Active request queue
- Adaptive timeout stuff
- Replies queue

By default, these resources will be allocated on all CPTs of global cfs_cpt_table for those most common used services of MDT and OST. However, it is not possible to allocate any number of CPTs to a service in the future (see "Initializer of ptlrpc service" in "Functional specification").

Graph-3 shows partitioned Lustre stack, although we expect all requests will be localized in one partition (Green lines), but it’s still possible that there are some cross-partition request (Red line):

- Not enough request-buffer in some partitions (should be rare.)
- User does not start service threads on all partitions.

With these cases, LND thread will steal buffers from other partitions and deliver cross-partition requests to service threads. This will degrade performance because inter-CPU data adds unnecessary overhead, but it is expected to be an improvement on the current situation because we can benefit from finer-grain locking in LNet and ptlrpc service. Locking in LNet and ptlrpc is judged to be one of the biggest SMP performance issue in Lustre.



Graph-3

Other thread-pool

There are other thread-pools which have will benefit from attention. For example, libcfs workitem schedulers (for non-blocking rehash), ptlrpcd, ptlrpc reply handler. Currently, all these thread-pools will create thread for each hyper-thread on the system. This behavior is sub-optimal because HTs are simply logic processors and because threads in different modules could be scheduled simultaneously.

With Project Apus thread numbers will be based on both number of CPU cores and CPU partitions, and decrease overall threads in Lustre.

Configurable parameters

Define configurable parameters available in the feature.

Default CPT table

By default, libcfs will create a global CPT-table (cfs_cpt_table) that can be referred by all other Lustre modules. This CPT table will cover all online CPUs on the system, number of CPTs (NCPT) is:

- NCPT must be power of 2

- If NCPU <= 4, NCPT=1
- Otherwise NCPT^2 <= NCPU < (NCPT+1)^2

For example, on 32 cores system libcfs will create 4 CPTs, and 64 cores system it will create 8 CPTs.

Configure CPT table by libcfs tunables

User also can define CPT-table by these two ways:

- Libcfs `npartitions=NUMBER`

Create a CPT-table with specified number of CPTs, all online CPUs will be evenly distributed into CPTs.

If `npartitions` is set to "1", it will turn off all CPU node affinity features.

- Libcfs `cpu_patterns=STRING`, examples:
  - cpu_patterns="0[0,4] 1[1,5] 2[2,6] 3[3,7]"

```
libcfs will create 4 CPTs and:
CPT0 contains CPU0 and CPU4, CPT1 contains CPU1 and CPU5,
CPT2 contains CPU2 and CPU6, CPT3 contains CPU3 and CPU7.
```

  - cpu_patterns="N 0[0-3] 1[4-7] 2[8-11] 3[12-15]"

```
libcfs will create 4 CPTs and:

CPT0 contains all CPUs in NUMA node[0-3]

CPT1 contains all CPUs in NUMA node[4-7]

CPT2 contains all CPUs in NUMA node[8-11]

CPT3 contains all CPUs in NUMA node[12-15]
```

- If `cpu_patterns` is provided, then `npartitions` will be ignored
- `cpu_patterns` does not have to cover all online CPUs. This has the effect that the user can make most Lustre server threads execute on a subset of all CPUs.

LNet tunables

By default, LND connections for each NI will be hashed to different CPTs by NID, LND schedulers on different CPTs can process messages for a same NI. However, a user can bind LNet interface on specified CPT with:

```
Lnet networks="o2ib0:0(ib0) o2ib1:1(ib1)"
```

The number after colon is CPT index. If user specified this number of a LNet NI, then all connections on that NI will only attached on LND schedulers running on that CPT. This feature could benefit NUMA IO performance in the future.

In previous example, all ib connections for `o2ib0` will be scheduled on CPT-0, all ib connections for `o2ib1` will only be scheduled on CPT-1.

API and Protocol Changes

LNet API changes

```
Int LNetEQAlloc(unsigned int count, lnet_eq_handler_t callback, …)
```

Providing both @count and @callback at the same time is discouraged. @count is only for LNetEQPoll and enqueue/dequeue for polling will be serialized by a spinlock which can hurt performance. A better mechanism in SMP environments is to set @count to 0 receive events from @callback.

```
LNetMEAttach(unsigned int portal, …, lnet_ins_pos_t pos, lnet_handle_me_t *handle);
```

@pos can take a new value: `LNET_INS_AFTER_CPU`. This has the effect that ME will be attached on local CPU partition only (the CPU partition that calling thread is running on). If @pos is `LNET_INS_AFTER`, then ME will always be attached on a same partition no matter which partition the caller is running on. In this mode, all LND threads will contend buffer on the partition and this is expected to degrade performance on SMP.

Open issues

Integrating the LNet Dynamic Config feature and this feature together. This project requires to make some changes to initializing/finalizing process of LNet, which might conflict with another project "LNet Dynamic Config".

Risks and Unknowns

- Performance on non-MDS systems

We expect this project can help on SMP scalability on all systems (MDS, OSS, clients), but we never benchmarked our prototype for non-MDS systems.

- The more tuning controls in a system, the more difficult it becomes to test all the possible permutations

This project adds new tunables to system, and it will not easy to test all code paths in our auto tests.