Interface Bonding

• A long-standing wish list item known under a variety of names:
  – Interface bonding
  – Channel bonding
  – Multi-rail
• Fujitsu implemented o2iblnd-level code and made it available to the community
• This proposal is a collaboration between SGI and Intel
• The goal is to land multi-rail support in Lustre
Why Multi-Rail?

SGI® UV™ 300: 32-socket NUMA System
SGI® UV™ 3000: 256-socket NUMA System
Design Constraints

Based on feedback for the Fujitsu code

- Mixed-version clusters
- Simple configuration
- Adaptable
- LNet-level implementation
Example Lustre Cluster
Mono-rail Single Fabric
LNets in a Single Fabric
Multi-rail Single Fabric

- Client
- Client
- UV
- o2ib0
- MGS
- OSS
- OSS
- OSS
- MGT
- MDT
- OST
- OST
- OST

©2015 SGI
Multi-rail Dual Fabric

Client

Client

UV

o2ib0

o2ib1

MGS

MGT

MDS

MDT

OST

OST

OST

OST

OSS

OSS

OSS

OSS
Mixed-Version Clusters

A Single Multi-Rail Node
Peer Version Discovery
A Single Multi-Rail Node

Client

Client

UV

o2ib0

MGS

MGT

MDS

MDT

OSS

OST

OST

OST

OSS

OSS

OSS

OST
Peer Version Discovery

• There is no LNet end-to-end versioning
• LND versioning does not work across LNet routers
• The LNet ping protocol can be used.

A set bit in \textit{lnet\_ping\_info\_t::pi\_features} indicates multi-rail capability.
Peer Version Discovery

A simple version discovery protocol:
1. LNet keeps track of all known peers
2. On first communication, do an LNet ping
3. The node now knows the peer version

The ping reply also contains a list of the interfaces of the peer. Can we use that?
Easy Configuration

Peer Interface Discovery
Configuring Interfaces on a Node
Dynamic Configuration
Peer Interface Discovery

- Peer Version Discovery gives a node the list of the peer’s interfaces.
- For the simple cases, this is all a node needs to know about a peer.
- The peer also needs to know the node’s interfaces.

Push the node’s interface list to the peer.
Peer Interface Discovery

The push would be like an LNet ping
• Except LNet ping uses LNetGet()
• The push uses LNetPut()

The push should be safe:
• A downrev LNet router can forward a push
• A downrev peer returns “protocol error”
Configuring Interfaces on a Node

How does a node know its own interfaces? Similar to current methods.

• LNet module options line
  – `networks=o2ib(ib0,ib1)`
  – `networks=o2ib(ib0[2],ib1[6])[2,6]`

• DLC uses the same syntax
  – It uses the same in-kernel parser
What about credits?

- Credits are assigned per interface.
- This applies to both local and peer credits.
- More interfaces – more credits.
- The defaults of tunables are unchanged.
Dynamic Configuration

Adding an interface:
1. Enable new interface
2. Push updated interface list to peers

Removing an interface:
1. Push updated interface list to peers
2. Disable existing interface
Adaptable

Interface Selection

Extended Routing

Additional Considerations
Interface Selection

Select a local-peer interface pair to send.

- Direct connection preferred
- LNet network type (anything but TCP)
- NUMA criteria
  - Memory locality
  - Process locality
- Local credits
- Peer credits
Routing Enhancements

Fabrics can have a complicated topology.

- Preferred point-to-point connections within an LNet

- Prefer an LNet over another (for a subset of its NIDs)
Extra Considerations

Try to be NUMA friendly.
• Nodes do not know each other’s topology
• An RPC is a request-response pair.
• Remember origin interface of request
• Prefer origin interface for response
Extra Considerations

On a send failure, a message can be resent on another local-remote interface pair, until all possibilities have been exhausted.

This adds some extra resiliency for network failures to Lustre.
Extra Considerations

Node failure introduces some corner cases.

• Reboot with downrev software
  – Upper layers (ptlrpc) do detect node failure
  – They can inform LNet so it can reset its state

• NID reuse by a different node
  – Node identity is now separate from NID
  – Special NIDs on the loopback network?
LNet-level Implementation

Implementation Notes
Implementation Notes

Datastructure changes

• Split \texttt{Inet\_ni} into \texttt{Inet\_Inet} and \texttt{Inet\_ni}
• Split \texttt{Inet\_peer} into \texttt{Inet\_peer} and \texttt{Inet\_peerni}
• Track preferred routes in \texttt{Inet\_net}
• Track preferred \texttt{Inet\_ni} in \texttt{Inet\_peerni} (derived from the routes info in \texttt{Inet\_net})
Implementation Notes

Use `LNET_NID_ANY` for the `self` parameter of `LNetGet()` and `LNetPut()` when sending an RPC request. This tells LNet to use whichever local-remote interface pair it seems most suitable.

Use the originator NID when sending an RPC response. This tells LNet that this particular local-remote pair is strongly preferred.
Implementation Notes

The Memory Descriptor can be extended with NUMA hints, to give LNet NUMA-specific information in selecting a suitable local interface.

Then LNet can select a remote interface for the peer that can be reached from the local interface.
Implementation Notes

1. Split *lnet_ni*
2. Local interface selection
3. Split *lnet_peer*
4. Ping on connect
5. Implement push
6. Peer interface selection
7. Resending on failure
8. Routing enhancements
Feedback & Discussion

Q&A

olaf@sgi.com