Lustre Configuration

February 11, 2008

1 Introduction

This High Level Design document introduces a new configuration and management interface for Lustre. Lustre configuration will become dramatically simpler and be true to the Unix spirit.

1.1 Overview of the mechanisms

We begin with a high level summary of the features that we discuss in this document. Lustre servers, metadata servers (**MDS**) and object storage servers (**OSS**), use file systems to store data and metadata. These file systems become so called object storage targets (OST) and metadata targets (MDT) in the global file system. Targets are (almost) invisible to users and of great importance to administrators. A Lustre format command (**mkfs.lustre)** is introduced that stores some additional configuration information on these file systems. To start services on these file systems, i.e. to activate the targets, a Unix mount command is issued. If a target is new it automatically joins the cluster, by contacting the configuration **management service**. Client nodes are also started with a mount command, and obtain file system configuration information from the same management service.

Management nodes can request all nodes to transparently update their configurations, using a versioned configuration schema. This provides the capability to dynamically add storage targets, migrate data and perform other storage management tasks. The schema provides a last full version and incremental updates to older versions of the configuration.

Lustre networking may be used by many file systems in a site, using multiple networks. In order to start Lustre client and services, nodes need to be aware how to traverse the networks, using portals routers, to reach management nodes. This is **bootstrap** information which cannot be fetched from the management service. While simple networks that only use TCP/IP or another single network type are configured automatically, more complex networks are configured via module parameters at module setup time.

Lustre bootstrap information, i.e. addresses of management nodes, are passed as arguments to formatting (on servers), and mount commands, and managed as persistent data in files like */etc/fstab*.

To enable centrally administered persistent updates to this information on all cluster nodes, clients and service nodes can use a **configuration cache file** which provides persistent storage of this information. The configuration cache is updated by these nodes in response to update requests from the management nodes.

1.2 Glossary

namespace The Lustre global file system namespace is mounted on all client and server nodes.

filesystem A lustre file system is internally made up of a collection of clustered metadata targets and object storage targets. Each target is offered by a file system on a metadata server (MDS) or object storage server (OSS)

kernel modules There will be one lustre.o module, this includes portals and the socknal. Other NALs will be available as separate modules.

servers (OSS/MDS) Are nodes offering services of the network. Service software is started at insmod time.

targets (ost/mdt) Define access to data on a file system. Targets are configured and then started by formatting a device as an ext3 file system for Lustre use and mounting the device as a Lustre file system on the servers. The namespace offered by the mount is the global namespace of Lustre.

a cluster is configured by configuring and starting a (failover pair of) MDS server(s) and by adding (pools of) OSS nodes and clustered MD servers

clients are configured and started with a mount command. The mount command names a management or MDS node and if necessary routers to reach this node. When clients have caches, the cache device is mounted as a Lustre file system to start the client. When a proxy is used the MDS is the proxy MDS.

echo clients/servers are configured through insmod.

management nodes For each file system a failover pair of nodes is designated as the master configuration management nodes. Normally this pair is a pair of MDS servers.

lustre networking allows clients and servers to communicate with each other. Lustre networking is configured via module parameters. The default parameters sets up simple TCP/IP networking.

network A network is a group of nodes that communicate directly with each other. It is how lustre networking represents a single cluster. Multiple networks are used to connect clusters together. Each network has a unique type and number (e.g. tcp3 elan). The number defaults to zero if it is omitted.

NID A NID is a lustre networking address, written <address>@<network>. The network defaults to 'tcp' if it is omitted. Every node has one NID for each network it is on.

routers are nodes on more than one network that have been configured to forward communications between these networks.

route table A route table lists networks and the routers that can reach them. Nodes that need to communicate with other nodes on different networks look up the route table to find a suitable router.

NAL stands for network abstraction layer. It is the software that implements a particular type of network.

2 Example

2.1 Basic Example

Modprobe parameters are not needed if the network used is TCP. If it is Elan or Voltaire infiniband we would use:

```
modprobe lustre.o networks=elan
modprobe lustre.o networks=vib
```

An MDS with a collection of OSS's with stripe count 3 and stride size 4M is configured and started as follows:

mds

```
mkfs.lustre --mdt --fsname=lustre1 --stripecount=3 --stripesize=4M \
/dev/hda5
mount.lustre /dev/hda5 /mnt/lustre
```

for each oss:

```
mkfs.lustre --ost --fsname=lustre1 --mgtnode=mgtnodename /dev/hda5
mount.lustre /dev/hda5 /mnt/lustre
```

client

```
mount.lustre mgtnodename:/lustre1 /mnt/lustre
```

The **mgmtnodename** would be the MDS host address. Mount.lustre is able to resolve IP hostnames to addresses. Numerous other options and configurations can be defined similarly.

To mount an Elan client:

```
mount 3@elan:/fsname /mnt/lustre
```

2.2 Complex example - an involved single global site wide file system

A site has a configuration to build a site wite global file system. The philosophy of the deployment has the following elements:

1. Three clusters are deployed. The clusters have fast networks (Elan) with high port costs. The clusters share a site-wide heavy duty MDS (to become an MDS cluster in future versions of Lustre), and each cluster *has* a storage pool, which while part of the global file system is preferred for that cluster. There is also a visualization cluster on the IP network that shares the file system.

2. The MDS nodes are attached to an infiniband network, which is connected with two portals routers per cluster to the cluster's Elan networks, elib[1-3]-[1-2]. The MDS's are also connected to the site wide IP network through 4 routers ipib[1-4]. The OSS storage pools reside on the IP network. They are named pool1, pool2, pool3 and they connect with 8 routers per cluster to the fast cluster networks. The routers are named elip[1-3]-[1-8], each of which have two gige interfaces and one Elan interface.

3. The IP network is tightly controlled and the OSS nodes flag the packets arriving from the OSS' *own* cluster N through elipN-[1-8] as coming from a trusted source, not requiring a GSS security context, the MDS nodes do the same with all packets arriving through the Elan-IB routers.

4. The first OSS pool cannot be read by cluster 2 and 3. On pool2 no file creations are possible by cluster 1, while cluster 3 cannot write pool2. Note that the permissions in this model have been chosen incremental, r < w < c (an alternate choice is possible).

(PLEASE INCLUDE A DRAWING)

Configuration definitions:

Lustre Networking is configured via module parameters. All nodes use the same settings. It is most convenient to set them in `/etc/modprobe.conf.`

Separate IP subnets have been used in this configuration so that similar nodes can share a subnet within the IP cluster. However the networking infrastructure allows full connectivity. IP interfaces have all been specified explicitly.

The IB cluster's IPoIB subnet is 192.168.0.0/24.

The Elan clusters have eip (IP over Elan) subnets of 132.6.[1-3].0/24. This IP addresses are only relevant to network selection.

The Elan routers on the IB cluster have IPoIB addresses 192.168.0.[1-6] with ElanIDs 1 and 2 on their respective Elan clusters.

The IP routers on the IB cluster have IPoIB addresses 192.168.0.[1-8], each with a single GigE NIC (eth1) at 134.9.0.[1-4].

The IP routers on the Elan clusters have ElanIDs [3-10]. They each have 2 GigE NICs (eth1, 134.9.1.[1-24] and eth2, 134.9.2.[1-24]).

```
options lnet 'ip2nets="vib           192.168.0.*    # IB cluster;\
                       elan1         132.6.1.*      # Elan cluster 1;\
                       elan2         132.6.2.*      # Elan cluster 2;\
                       elan3         132.6.3.*      # Elan cluster 3;\
                       tcp(eth1)     134.9.0.[1-4]  # IB/TCP routers;\
                       tcp(eth1,eth2) 134.9.1.[1-24] # Elan/TCP routers;\
                       tcp           *.*.*.*        # The rest of the world"'\
              'routes="tcp   192.168.0.[1-8]@vib     # IB cluster routers;\
                       elan1 192.168.0.[1-2]@vib;\
                       elan2 192.168.0.[3-4]@vib;\
                       elan3 192.168.0.[5-6]@vib;\
                       vib   134.9.0.[1-4]@tcp       # TCP routers;\
                       elan1 134.9.1.[1-8]@tcp;\
                       elan2 134.9.2.[9-16]@tcp;\
                       elan3 134.9.3.[17-24]@tcp;\
                       vib   [1-2]@elan[1-3]         # Elan[1-3] routers;\
                       tcp   [3-10]@elan[1-3];"'
```

Note that the last two routes suffice for all 3 elan clusters because they all use the same ElanIDs for similar routers.

OSS/MDS-pools are configured with the lformat command.

MDS

```
mkfs.lustre --mdt --fsname=swgf                      \
        --ostpool=pool1 --stripecnt=4 --stripesz=4M \
            --access=c@elan1                        \
        --ostpool=pool2 --stripecnt=5 --stripesz=5M \
            --access=w@elan1,c@elan2,r@elan3        \
        --ostpool=pool3 --stripecnt=6 --stripesz=7M \
            --access=c@elan3,c@elan2,c@elan1        \
        --failover=mds2@vib --clumanager            \
          --securenet=elib[1-3]-[0-2] /dev/sdaQ
```

OSS pool X

```
mkfs.lustre --ost --fsname=swgfs --ostpool=poolX --heartbeat \
        --failover=ossY --mgtnode=mds[1-2]@4          \
        --securenet=elipX-[1-8] /dev/sdaP
```

3 Lustre Networking (LNET)

3.1 LND

Lustre networking provides communication services to lustre clients and servers over a standard interface. This interface abstracts the actual network hardware being used, its addressing schemes and transport methods. The software that understands specific hardware is called the LND which stands for "lustre network driver". Each network type has its own LND which is loaded on demand as lustre networking is configured.

3.2 NID

Lustre servers are addressed by a lustre network ID, called a NID. A NID consists of a network and an address within that network written as <address>[@<network>].

The <network> part of a nid consists of a network type (which determines the LND) and a number (e.g. "elan3" or "tcp2") which distinguishes different instances of the same network type. The network number can be omitted and it defaults to zero (i.e. "tcp" and "tcp0" mean the same network). If the whole '@<network>' part of a NID is omitted, it defaults to 'tcp'.

The <address> part of a nid is a expressed in the "natural" syntax for the given network type. For example addresses on an Elan network are numbered from 0 - (nnodes - 1), so the node with ElanID 2 has the NID 2@elan. Similarly 134.9.37.164@tcp is a NID on a TCP network. Some lustre utilities allow the IP address within a NID to be specified using a hostname (e.g. tdev3@tcp), however this is not permitted in module parameters.

3.3 Networks

All nodes that share the same network number can communicate with each other. Typically, a single cluster is spanned by a single Lustre network; so no further network configuration is required for cluster-local communications. Sites with only a single cluster are therefore very simple to set up.

Multiple networks are used when several clusters all want to share the same lustre file systems. Nodes that are members of more than one network provide the connectivity. Services on these nodes can serve their local networks directly, but they can also be set up as routers. Routers forward communications between their local networks. They allow clients in the *interior* of one network to access servers in the *interior* of another network.

Nodes that need to communicate with non-local networks look up the destination network in a route table. This table is set up as part of network configuration. It lists accessible networks and the routers that can reach them.

3.4 Servers with Mulitple NIDs

A node's connection to a lustre network is called a lustre network interface (LNI). Each LNI has its own NID.

Clients of a Lustre server with multiple NIDs must use the correct NID to communicate with the server to ensure that their communications are routed properly. This occurs automatically for all OSTs once the connection to the MDS so it's only really important to know which MDS NID to use in the mount command.

3.5 Configuration

Lustre networking is configured at module load time by setting module parameters. These parameters have "sensible" defaults so that lustre can be used "out-of-the-box" with no configuration in simple situations.

The same set of configuration parameters can be specified for all nodes in a given site configuration; redundant configuration information is simply discarded.

Under linux 2.6, the lustre configuration network parameters can be viewed under /sys/modules; generic parameters under 'lnet' ('lustre' when lnet is merged into lustre) and LND specific parameters under the corresponding LND's name.

Under linux 2.4, sysfs is not available, but the LND-specific parameters are accessible via equivalent paths under /proc.

In the descriptions of network parameters below, the text in brackets following the parameter name shows its default value and a "W" if the parameter is writeable via sysfs/procfs. Changes to writeable parameters have immediate effect on a running system.

3.5.1 Network Topology

The network topology module parameters determine which networks a node should join, whether it should route between these networks and how it communicates with non-local networks.

ip2nets ("") is a string that lists networks, each with a set of IP address ranges.
It has the following syntax...

```
<ip2nets>    :== <net-match> [ <comment> ] { <net-sep> <net-match> }
<net-match>  :== [ <w> ] <net-spec> <w> <ip-range> { <w> <ip-range> } [ <w> ]
<net-spec>   :== <network> [ "(" <interface-list> ")" ]
<network>    :== <nettype> [ <number> ]
<nettype>    :== "tcp" | "elan" | "openib" | ...
<iface-list> :== <interface> [ "," <iface-list> ]
<ip-range>   :== <r-expr> "." <r-expr> "." <r-expr> "." <r-expr>
<r-expr>     :== <number> | "*" | "[" <r-list> "]"
<r-list>     :== <range> [ "," <r-list> ]
```

```
<range>       :== <number> [ "-" <number> [ "/" <number>  ] ]
<comment      :== "#" { <non-net-sep-chars> }
<net-sep>     :== ";" | "\n"
<w>           :== <whitespace-chars> { <whitespace-chars> }
```

The <net-spec> contains enough information to identify the network uniquely and load an appropriate LND. The LND determines the missing "address-within-network" part of the NID based on the interfaces it can use.

The optional <iface-list> specifies which hardware interface the network can use. LNDs that do not support the <iface-list> syntax can not be configured to use particular interfaces just use "what's there". Only a single instance of these LNDs can exist on a node at any time, and the <iface-list> must be omitted.

The linux kernel version of the TCP LND (the socknal) does support the <iface-list> syntax. It uses the first interface specified to determine the local address part of the NID, and it tries to balance network traffic over all its interfaces. If no interfaces are specified, the socklnd uses all available interfaces, so it's important to specify all the interfaces to use, if there are any interfaces it should not use. [1]

The <net-match> entries are scanned in the order declared to see if one of the node's IP addresses matches one of the <ip-range> expressions. If there is a match, the <net-spec> specifies the network to instantiate. Note that it is the first match for a particular network that counts. This can be used to simplify the match expression for the general case by placing it after the special cases. For example..

- `ip2nets="tcp(eth1,eth2) 134.32.1.[4-10/2]; tcp(eth1) *.*.*.*"` says that 4 nodes on the 134.32.1.* network have 2 interfaces but all the rest have 1.
- `ip2nets="vib 192.168.0.*; tcp(eth2) 192.168.0.[1,7,4,12]"` describes an IB cluster with 4 nodes at irregular IP addresses that also have IP interfaces and could be used as routers.

Note that match-all expressions (e.g. *.*.*.*) effectively mask all other <net-match> entries specified after them; they should be used with caution.

networks ("tcp") is an alternative to "ip2nets" which can be used to specify which networks to instantiate explicitly. The syntax is a simple comma separated list of <net-spec>s (see above). The default is only used if neither 'ip2nets' nor 'networks' is specified

[1] Consider a node on the "edge" of an Infiniband network, with a low bandwidth management ethernet (eth0), IP over IB configured (ipoib0), and a pair of GigE NICs (eth1,eth2) providing off-cluster connectivity. This node should be configured with 'networks="vib,tcp(eth1,eth2)"' to ensure that the socknal ignores the management ethernet and IPoIB.

routes ("") is a string that lists networks and the NIDs of routers that forward to them. It has the following syntax...

```
<routes>    :== <route> [ <comment> ] { <route-sep> <route> }
<route>     :== [ <w> ] <net> [ <w> <hopcount> ] <w> <nid> { <w> <nid> } [ <w> ]
<comment    :== "#" { <non-route-sep-chars> }
<w>         :== <whitespace-chars> { <whitespace-chars> }
<route-sep> :== ";" | "\n"
<hopcount>  :== <number>
```

A simple but powerful expansion syntax is provided, both for target networks and router NIDs as follows...

```
<expansion>     :== "[" <entry> { "," <entry> } "]"
<entry>         :== <numeric range> | <non-numeric item>
<numeric range> :== <number> [ "-" <number> [ "/" <number> ] ]
```

The expansion is a list enclosed in square brackets. Numeric items in the list may be a single number, a contiguous range of numbers, or a strided range of numbers. For example...

- `'routes="elan 192.168.1.[22-24]@tcp"'` says that network elan0 is adjacent (hopcount defaults to 1), and is accessible via 3 routers on the tcp network (192.168.1.22@tcp, 192.168.1.23@tcp and 192.168.24@tcp).
- `'routes="[tcp,vib] 2 [8-14/2]@elan"'` says that 2 networks (tcp0 and vib0) are accessible through 4 routers (8@elan, 10@elan, 12@elan and 14@elan). The hopcount of 2 means that traffic to both these networks will be traverse 2 routers; first one of the routers specified in this entry, then one more.

The default, being the empty string, means that non-local networks are unreachable.

Routing (i.e. forwarding betwen networks) is enabled If a route includes a router with a NID that matches one of the node's own NIDs and routing has not been disabled explicitly (see "forwarding" below).

Duplicate entries, entries that specify a local network, and entries that specify routers on a non-local network are ignored. Equivalent entries are resolved in favour of the route with the shorter hopcount. The hopcount, if omitted, defaults to 1 (i.e. the remote network is adjacent).

forwarding ("") is a string that can be set either to "enabled" or "disabled" for explicit control of whether this nod should act as a router, forwarding communications between all local networks.

3.5.2 Routing and Credits

LNET implements a credit flow control system to ensure that communications can continue to flow through routers in the presence of congestion and/or dead peers. The credit system ensures that no single peer can consume all the router's resources.

tiny_router_buffers sets the number of zero-payload router buffers available for forwarding.

small_router_buffers sets the number of single-page router buffers available for forwarding.

large_router_buffers sets the number of maximum payload router buffers available for fowarding.

LND credits controls how many concurrent sends in total an instance of the LND will support before causing communications to block. Many LNDs also have a module parameter (e.g. "ntx") that sets the number of pre-allocated message descriptors, and 'credits' should be sufficiently less than this to allow for descriptor allocation in response to RDMA requests.

LND peer_credits controls how many concurrent sends to a particular peer an instance of the LND will support before causing communications to that peer to block. When 'peer_credits' is small compared with 'credits', many peers have to be unresponsive before "healthy" peers start to be starved of resources.

This also controls how many router buffers (of any size) any particular peer may reserve. When this limit has been reached, the router will block further buffer requests from this peer.

3.5.3 Acceptor

The acceptor is a TCP/IP service that some NALs use to establish communications. If it is required by a local network and it has not been disabled, the acceptor listens on a single port for connection requests which it redirects to the appropriate local network. The acceptor is configured by the following module parameters.

accept ("secure") is a string that can be set to any of the following values.

- **secure** - accept connections only from reserved TCP ports ($<$ 1023).
- **all** - accept connections from any TCP port.
- **none** - do not run the acceptor

accept_port (988) is the port number on which the acceptor should listen for connection requests. **All** nodes in a site configuration that require an acceptor must use the same port.

accept_backlog (127) is the maximum length that the queue of pending connections may grow to (see listen(2)).

accept_timeout (5,W) is the maximum time in seconds the acceptor is allowed to block while communicating with a peer.

accept_proto_version is the version of the acceptor protocol that should be used by outgoing connection requests. It defaults to the most recent acceptor protocol version, but it may be set to the previous version to allows the node to initiate connections with nodes that only understand that version of the acceptor protocol. The acceptor can, with some restrictions, handle either version (i.e. it can accept connections from both 'old' and 'new' peers). For the current version of the acceptor protocol (version 1), the acceptor is compatible with old peers if it is only required by a single local network.

3.5.4 Portals Compatibility

LNET can inter-operate with lustre-portals in single-network configurations to allow phased upgrades. This is controlled by the 'portals_compatibility' module parameter as follows...

portals_compatibility ("none") is a string that can have any of the following values.

- **strong** - compatible with portals and with LNET running either strong or weak portals compatibility mode. Set this while any other nodes are still running portals.
- **weak** - Not compatible with portals, but compatible with LNET running in any mode.
- **none** - Not compatible with portals, or with LNET running in strong portals compatibility mode.

In summary, when first introducing LNET to a site running portals, all LNET nodes should be set to "strong" compatibility. When the last portals node has been replaces by LNET, LNET nodes may be rebooted in "weak" compatibility mode. When all LNET nodes are running in "weak" compatibility mode, LNET can be booted as normal.

3.5.5 Other generic parameters

config_on_load (1) is a boolean that determines whether lustre networking should configure itself at module load time (set) or on first use (clear).

3.5.6 Kernel TCP/IP NAL

The socknal is connection based and uses the acceptor to establish communications via sockets with its peers.

It supports multiple instances and load balances dynamically over multiple interfaces. If no interfaces are specified by the "networks" module parameter, all non-loopback IP interfaces are used.

The address-within-network is determined by the address of the first IP interface an instance of the socknal is using.

Changes to parameters marked with a 'Wc" only have effect when connections are established. Existing connections are not affected by changes to them.

timeout (50,W) is the time in seconds that communications may be stalled before the NAL will complete them with failure.

credits (256) the maximum number of concurrent sends.

peer_credits (8) the maximum number of concurrent sends to any individual peer. Also the maximum number of router buffers any particular peer can use concurrently.

nconnds (4) sets the number of connection daemons.

min_reconnectms (1000,W) is the minimum connection retry interval in milliseconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnectms'.

max_reconnectms (60000,W) is the maximum connection retry interval in milliseconds.

eager_ack (0 on linux, 1 on darwin,W) is a boolean that determines whether the socknal should attempt to flush sends on message boundaries.

typed_conns (1,Wc) is a boolean that determines whether the socknal should use different sockets for different types of message. When clear, all communication with a particular peer takes place on the same socket. Otherwise separate sockets are used for bulk sends, bulk receives and everything else.

min_bulk (1024,W) determines when a message is considered "bulk".

buffer_size (8388608,Wc) sets the socket buffer size. Note that changes to this parameter may be **rendered ineffective** by other system-imposed limits (e.g. /proc/sys/net/core/wmem_max etc).

nagle (0,Wc) is a boolean that determines if nagle should be enabled. It should never be set in production systems.

keepalive_idle (30,Wc) is the time in seconds that a socket can remain idle before a keepalive probe is sent. 0 disables keepalives

keepalive_intvl (2,Wc) is the time in seconds to repeat unanswered keepalive probes. 0 disables keepalives.

keepalive_count (10,Wc) is the number of unanswered keepalive probes before pronouncing socket (hence peer) death.

irq_affinity (1,Wc) is a boolean that determines whether to enable IRQ affinity. When set, the socknal attempts to maximize performance by handling device interrupts and data movement for particular (hardware) interfaces on particular CPUs. This option is not available on all platforms.

zc_min_frag (2048,W) determines the minimum message fragment that should be considered for zero-copy sends. Increasing it above the platform's PAGE_SIZE will disable all zero copy sends. This option is not available on all platforms.

3.5.7 QSW NAL

The qswnal is connectionless, therefore it does not need the acceptor.

It is limited to a single instance, which uses all Elan "rails" that are present and load balances dynamically over them.

The address-with-network is the node's Elan ID. A specific interface cannot be selected in the "networks" module parameter.

tx_maxcontig (1024) is a integer that specifies the maximum message payload in bytes to copy into a pre-mapped transmit buffer.

ntxmsgs (256) is the maximum number of message descriptors for sending messages and mapping RDMA buffers.

credits (128) is the maximum number of concurrent sends.

peer_credits (8) is the maxumum number of concurrent sends to a single peer and also the maximum number of router buffers that can be used at any time by a single peer.

nrxmsg_small (256) is the number of "small" receive buffers to post (typically everything apart from bulk data).

ep_envelopes_small (2048) is the number of message envelopes to reserve for the "small" receive buffer queue. This determines a breakpoint in the number of concurrent senders. Below this number, communication attempts are queued, but above this number, the pre-allocated envelope queue will fill, causing senders to back off and retry. This can have the unfortunate side effect of starving arbitrary senders, who continually find the envelope queue is full when they retry. This parameter should therefore be increased if envelope queue overflow is suspected.

nrxmsg_large (64) is the number of "large" receive buffers to post.

ep_envelopes_large (256) is the number of message envelopes to reserve for the “large” receive buffer queue. See “ep_envelopes_small” above for a further description of message envelopes.

optimized_puts (32768,W) is the smallest bulk payload that will be RDMA-ed.

optimized_gets (1,W) is the smallest non-routed GET that will be RDMA-ed.

3.5.8 RapidArray NAL

The ranal is connection-based and uses the acceptor to establish connections with its peers.

It is limited to a single instance, which uses all (both) RapidArray devices present. It load balances over them using the XOR of the source and destination NIDs to determine which device to use for any communication.

The address-within-network is determined by the address of the single IP interface that may be specified by the “networks” module parameter. If this is omitted, the first non-loopback IP interface that is up, is used instead.

n_connd (4) sets the number of connection daemons.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (30,W) is the time in seconds that communications may be stalled before the NAL will complete them with failure

ntx (256) is the number of message descriptors for sending and mapping RDMA buffers.

credits (128) is the maximum number of concurrent sends.

peer_credits (32) is the maximum number of concurrent sends to a single peer and also the maximum number of router buffers that a single peer can use concurrently.

fma_cq_size (8192) is the number of entries in the RapidArray FMA completion queue to allocate. It should be increased if the ranal starts to issue warnings that the FMA CQ has overflowed. This is only a performance issue.

max_immediate (2048,W) is the size in bytes of the smallest message that will be RDMA-ed, rather than being included as immediate data in an FMA. All messages over 6912 bytes must be RDMA-ed (FMA limit).

3.5.9 Voltaire NAL

The vibnal is connection based, establishing reliable queue-pairs over infiniband with its peers. It does not use the acceptor for this.

It is limited to a single instance, which uses a single HCA that can be specified via the "networks" module parameter. It this is omitted, it uses the first HCA in numerical order it can open.

The address-within-network is determined by the IPoIB interface corresponding to the HCA used.

Changes to parameters marked with a 'Wc" only have effect when connections are established. Existing connections are not affected by changes to them.

service_number (0x11b9a2) is the fixed IB service number on which the NAL listens for incoming connection requests. Note that **all** instances of the vibnal on the same network must have the same setting for this parameter.

arp_retries (3,W) is the number of times the NAL will retry ARP while it establishes communications with a peer.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (50,W) is the time in seconds that communications may be stalled before the NAL will complete them with failure.

ntx (256) is the number of message descriptors used for sending or mapping RDMA buffers.

credits (128) is the maximum number of concurrent sends.

peer_credits (8) is the maximum number of concurrent sends to a single peer and also the maximum number of router buffers that any single peer can use concurrently.

concurrent_peers (1152) is the maximum number of queue pairs, and therefore the maximum number of peers that the instance of the NAL may communicate with.

hca_basename ("InfiniHost") is used to construct HCA device names by appending the device number.

ipif_basename ("ipoib") is used to construct IPoIB interface names by appending the same device number as is used to generate the HCA device name.

local_ack_timeout (0x12,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

retry_cnt (7,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

rnr_cnt (6,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

rnr_nak_timer (0x10,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

fmr_remaps (1000) controls how often FMR mappings may be reused before they must be unmapped. It should not be changed from the default unless advised.

cksum (0,W) is a boolean that determines whether messages (NB not RDMAs) should be checksummed. This is a diagnostic feature that should not be enabled normally.

3.5.10 OpenIB NAL

The openibnal is connection based and uses the acceptor to establish reliable queue-pairs over infiniband with its peers.

It is limited to a single instance that uses only IB device '0'.

The address-within-network is determined by the address of the single IP interface that may be specified by the "networks" module parameter. If this is omitted, the first non-loopback IP interface that is up, is used instead. It uses the acceptor to establish connections with its peers.

n_connd (4) sets the number of connection daemons. The default is 4.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (50,W) is the time in seconds that communications may be stalled before the NAL will complete them with failure.

ntx (384) is the number of message descriptors for sending or mapping RDMA buffers.

credits (256) is the maximum number of concurrent sends.

peer_credits (16) is the maximum number of concurrent sends to any individual peer and also the maximum number of router buffers that any individual peer can use concurrently.

concurrent_peers (1024) is the maximum number of queue pairs, and therefore the maximum number of peers that the instance of the NAL may communicate with.

cksum (0,W) is a boolean that determines whether messages (NB not RDMAs) should be checksummed. This is a diagnostic feature that should not be enabled normally.

3.6 Route tracking.

On TCP networks it is desirable for Lustre to have knowledge of the IP address from which incoming packets were sent out before they reached the destination. This can be the source node in case of a direct connection or it can be the last router in a routed Lustre network.

The events associated with packet deliver carry a field to encode this IP address.

4 Lustre Configuration

4.1 Management nodes

This section considers the features in which management nodes play a role. The management nodes hold the descriptors for OST's and MDT's, serve configuration data to clients when these perform a mount of a Lustre file system, and the management nodes propagate updates of configuration elements to all clients.

The management nodes can store the configuration in a partition solely used for configuration data, in a partition which is used as an MDT service partition or in a file which is formatted as a file system.

The management service can be a failover service relying on shared storage, as is done for MDS and OSS targets.

4.2 Initial cluster configuration

The proposal of this design is to define formatting commands and insmod commands that configure a cluster completely. This means that the following sequence configures and starts a cluster:

1. Correctly insmod lustre.o to set up Portals routing and network definitions.
2. Format and start the management node, normally the primary MDT
3. Format and start each OST. The first time an OST (or an additional clustered MDT) is started by mounting it, the node communicates with the management node which modifies the configuration of the Lustre cluster through the addition the OST/MDT.

4. Restart the management node which exports a configuration including all new OST and MDT targets
5. Start all clients with a mount command

Initial implementation After the initial startup in the given order, nodes can be started in any order. The initial implementation will target exactly the sequence above and random starting order thereafter.

Target implementation The next implementation step is to arrange online addition of OST and MDT targets. At this point, step 4 above will not be necessary and the order of startup can be relaxed, clients can be started any time.

The final implementation step is to arrange online addition and removal of routers (subject to interest from our routing customers).

4.2.1 File System Startup

The file system can be started by issuing insmod commans on routers and mount commands on all other nodes, in any order. If redundant Portals routers are available startup will use them without significant delays. If some targets are not available startup will proceed without delays and access the nodes when they become available. Warnings will be issued to notify operators of targets that have failed to start.

4.2.2 File System Management

Dynamic, online OST addition (as well as removal of OST and MDT addition to a cluster) is desirable. For this typically the primary MDT nodes will double as management nodes that can communicate with clients and OSS's in the filesystem to request these do a configuration refresh. Each file system has its own management nodes.

A configuration refresh request can be handled by a client as follows:

1. **Immediate handling** results in transparent updates to the configuration. All system calls complete normally while a configuration changes.
2. Handling upon **reconnect** allows a client to fetch all the changes to the configuration that were made since it last connected. This way of handling is not transparent, but the semantics are that of an eviction.
3. Handling upon **remount** is simpler. In this case the client simply fetches the final configuration of the cluster and uses it.

The dynamic OST addition is based on mechanisms similar to those developed for Hendrix.

Initial implementation Additional targets can be incorporated by bringing clients down and repeating step 3-5. OST targets can be administratively removed by an update command on the MDS followed by unmounting and re-mounting clients.

Target implementation Step 3 can be repeated at any time to add new targets. Dynamic updates on the management node can define new pools and can remove OST's.

4.2.3 Failure management

All Lustre services are failover, including the management service. To start clients, the management service must be running. To add targets the management service must be running. If the management service is not running or not reachable due to network failures, client startup and target addition commands will succeed when the service becomes available.

OST targets have an involved failure model which includes handling the case of unrecoverable damage and ultimate removal of the target.

1. **Normal operation** of an OST involves clients waiting for target recovery through failover or reboot. By default these waits become interruptible after a timeout period. In a future release a maximal value for the wait can be configured after which errors are returned to waiting system calls.

2. An OST which has failed and will require time to possibly can be marked administratively as **failed.** Cients trying to access a failed OST will get immediate errors when trying to read or write data to objects on the OST. New object creations will avoid the failed OST, to render a file system nominally useful.

3. An OST which cannot be repaired can be **removed** from the cluster. When this is done system calls will mask failures to reach missing data, i.e. data on the removed OST and treat the missing data as a sparse region in the file. The option has dubious semantics must be used with extreme prejudice

Initial implementation The commands to fail and remove an OST commands are issued on all clients and MDS nodes.

Target implementation The commands can be issued on any node and are propagated throughout the file system by the management nodes.

4.2.4 Configuration update protocol

All nodes in the tree take a configuration lock at their parent node, changes in part of the configuration are announced through a lock callback. These locks are locks on an auxiliary resource related to configuration and not related to file system locks. However, the standard Lustre device (formally *obd device)* infrastructure is used to manage the locks. The locks are primarily used because of their convenient callback infrastructure and automatic removal when nodes stop pinging.

A recursive tree update protocol allows changes in configurations to be propagated.

The primary purpose of the update protocol is to provide information about new servers to clients and eliminate long-term non-responsive routers and servers from their configuration, such as declaring an OST failed. The protocol gives nodes enough time to quiesce, without incurring timeouts and update their configuration. Connection imports will register their configuration versions on all server exports and servers can be configured to evict imports that do not have up-to-date configurations running. A newly connecting client must fetch the most up to date configuration.

The nature of the protoocol is simple. All nodes acquire a configuration lock upon mount and fetch the configuration information. When the lock callback issued, nodes have an opportunity to flush buffers, drop the lock and fetch the updated protocol. Nodes that do not respond are evicted, which allows them to reconnect later, still semi-transparently. Eviction involves the loss of cached data and can lead to application errors.

Server nodes start from information stored in their disk file system. A new server contacts the master management node and informs it of its addition to the cluster. In the distant future, management nodes can be dynamically changed through a slightly more involved handshake.

4.2.5 Topology

! no more route managers (fttb) !

Management nodes form a spanning tree, the root is formed by route managers. A Lustre cluster may have a routing management tree and multiple file system management trees, which start at level one of the route management tree. Changes are only made at the root of management trees.

5 Configuration Utility Specification

5.1 mkfs.lustre

```
mkfs.lustre (--ost|--mdt|--mgmt) [--configmgr] [--netmgr]\
--fsname=<fsname> --targetid=<string>\
[--mgtnode=<hostdesc>[,<alt hostdesc>]]\
[--failover=<hostdesc> (--clumanager|--heartbeat)]\
[--stripecount=<cnt> --stripesize=<size>]\
[--stripeindex=<index>]\
[--ostpool=<poolname> --access=<access desc> --stripecount=<cnt>\
--stripesize=<size>]\
[--smfsopts <smfs options>]\
    [--ext3opts <ext3 options>]\
    (--loop <filename>|<partition>) \
    device
```

5.1.1 Options

ost|mdt This file system is a metadata or object target

[–configmgr] This target offers configuration management services for this file system. This means this node can be used in a client mount command to obtain client configuration logs from this target.

[–netmgr] This target offers ping, trap and other network management services used by Lustre for this file system. This flag is typically used by the primary MDS target and also by the client file system that is mounted on the failover server for this file system.

[–configdev=<device|file>] Use the device or file to store the configuration data instead of the target partition. This is mandatory if the target file system is a memory based file system like tmpfs.

[–failover=<address> (–clumanager|–hearbeat)] Use the server at <address> as a failover node for this target. Use clumanager or heartbeat to configure the service for this target. On the failover node a client file system will be mounted where the target mounts.

–fsname=<name> This parameter is a name of not more than 8 characters for the file system. Mkfs.lustre labels file systems and uses the fsname and targetid to provide labels for OSS and MDS file systems.

[–targetid=<string>] If the target id argument is given, mkfs.lustre labels the partition with a logical name, and requests the MDS to use this name, prefixed with the fsname to name the target on this server.
The recommended (and default) targetid's are **-ost<index>** and **-mdt<index>.** With this mkfs.lustre writes labels on the target partitions as ***<fsname>*-ost<index>** or **<fsname>-mdt<index>.**
Here index is a hex integer of up 0xfffe (index 0xffff is reserved). This naming requires the administrator to query the Lustre file system about the indices already in use. It provides defense against confusion arising from device names, which sometimes change if devices are added or removed from systems.
If this argument is not given, no label is written and the MDT will name the target as <fsname>_<ost|mdt><index>.

[–nolabels] The disk partition on this node is not labeled. The use of this option is not recommended.

[–stripeindex=<idx>] Add the server partition at index <idx> in the LOV or LMV descriptor. Index can be an arbitrary integer smaller than the maximum OST target number (XXX is this true or necessary?) Stripes normally go over OST's of consecutive indexes and it is desirable to avoid this if multiple targets have consecutive stripes on one server. This option can be used to influence this. If the index is already taken, the mount will fail and the MDS will issue a warning message.

–ostpool=<name> For OST's this OST will be added in the pool as the next OST in the pool. On MDT's this declares a new OST pool. It is typically followed by a stripecount and stripesize argument, which then applies to this pool. If a poolname is declared on an MDT, no default stripe parameters can be given and each pool much be given stripe parameters.

–stripecount=<count> Use <count> stripes in on the OST's or in the pool named by the preceeding poolname command.

–stripesize=<bytes> Bytes to use per stripe.

–loop the device to be formatted is a regular file and will be used through a loop device.

5.1.2 Description

Format a file system for use as an OST or MDT target.

5.2 mount.lustre

```
mount.lustre -o
[maxwait=<secs>] [timeout=<secs>]
[uluid=<uid>] [ulgid=<gid>]
<mds hostdesc>[,<alt mds hostdesc]:/<fsname> <mtpt>
```

5.2.1 Description

Mount.lustre starts services for targets. It must be issued to make targets available in the Lustre cluster. The command is normally invoked by the service management features of clumanager or heartbeat, and not by the operator, to ensure the service is started on exactly one node.

5.3 umount.lustre

5.4 lfs listtargets

```
lfs listtargets <mtpt>
```

Show a list of all target names, primary and failover server addresses associated with the file system.

5.5 lfs export

```
lfs export stop <mdt|ost mtpt>
lfs export failover <mdt|ost mtpt>
lfs export fsname=<fsname> [root_squash,no_setuid,net-desc:perm]
lfs export setfailover --host=<host addr> --tgt=<target name>\
   <lustre mtpt>
lfs export addpool --ostpool=<pool3name> --stripecnt=<cnt>\
   --stripesz=<size> <lustre mtpt>
lfs export delpool --ostpool=<pool3name> <lustre mtpt>
lfs export fail ost=<ost name> <fs mtpt>
lfs export remove osshost=<oss hostdesc>\
   osspart=<oss partition> <fs mtpt>
```

5.5.1 Description

stop	Stop the service on the target associated with the mountpoint. Clients will experience failures and cannot recover systems calls in progress upon reconnect. This is an online command.
failover	Stop the target but retain export related information that allows clients to reconnect to a failover or rebooted instance of this target. This is an online command.
setconfigmgr	Set the management that this target uses to the <host addr> or provide a primary and secondary management node as <host addr>,<failover host addr>.
setnetmgr	Set the network management node for this target, as for setmgr.
setfailover	Set the failover host for the target to the given <host addr>
addpool	Add a pool with <poolname> and the specified striping information for this pool.
delpool	Remove the pool from the configuration, only possible if the pool has no more targets registered
fail	Put the named ost on the file system in failed mode, i.e. it is not available right now and errors are returned without attempting to contact the OST.
remove	Remove the OST from the configuration permanently. Now when reading, missing data is returned as 0's and upon writing, new objects are created to replace missing objects.
addost	Add an OST the configuration. This command only updates the configuration data on the management node.

addmdt	Add an MDT the configuration. This command only updates the configuration data on the management node.

The commands that are not marked as online can be issued to a Lustre mounted file system or to that same file system mounted as ext3.

5.6 lfs migrate

```
lfs migrate --srccost=<source ost name>\
    --dstost=<destination ost name> <fs mtpt>
```

5.6.1 Description

XXX NOTE this needs a pool variant that allows re-striping of data while it is being migrated.

Migrate the data from the source OST to the target OST.

6 Operational Scenarios

Note that mount commands can be given in any order, after the system has been mounted the first time. Below a '\' (backslash) continuation character is used to indicate that commands are too long to fit on one text line.

In all examples below the management node is *the MDS*. To be precise the management service is started as the initial part of the startup of the primary MDT.

All targets that are configured for failover must have some kind of shared storage among two server nodes.

6.1 IP network, Single MDS, single OST, no failover

mds

```
mkfs.lustre  --mdt --fsname=<fsname> --tgtid=mdt<idx>\
    <mds partition>
mount.lustre <mds partition> <mds mtpt>
```

oss

```
mkfs.lustre --ost --fsname=<fsname> --tgtid=ost<idx>\
    --mgtnode=<hostdesc> <oss partition>
mount.lustre <oss partition> <oss mtpt>
```

In a secure environment, the OSS connection to the MDS is authorized on the grounds of a Kerberos service ticket for the MDS and a kerberos principal ticket for the OSS. The OSS principal has OSS addition priviliges on the MDS.

client

```
        mount.lustre <hostdesc>:/<fsname> <cli mtpt>
```

6.2 IP Network, one MDS striped collection of OSS

mds

```
        mkfs.lustre --mdt --fsname=<fsname> --stripecount=<cnt>\
            --stripesize=<size> <mds partition>
        mount.lustre <mds partition> <mds mtpt>
```

for each oss:

```
        mkfs.lustre --ost --fsname=<fsname>\
            --mgtnode=<hostdesc> <oss partition>
        mount.lustre <oss partition> <oss mtpt>
```

client

```
        mount.lustre <hostdesc>:/<fsname> <cli mtpt>
```

6.3 IP Network, failover MDS

For failover, storage holding target data must must be available as shared storage to failover server nodes. Failover nodes are statically configured as mount options.

mds

```
    mkfs.lustre --mdt --fsname=<fsname>\
       --failover=<failover mds hostdesc> (--clumanager|--heartbeat)\
       <mds partition>
    mount.lustre <mds partition> <mds mtpt>
```

oss

```
        mkfs.lustre --ost --fsname=<fsname>\
            --mgtnode=<mds hostdesc>,<failover mds hostdesc>\
            <oss partition>
        mount.lustre <oss partition> <oss mtpt>
```

client

```
        mount.lustre <mds hostdesc>[,<failover mds hostdesc>]:/<fsname>\
            <cli mtpt>
```

6.4 IP Network, failover MDS & OSS

mds

```
mkfs.lustre --mdt --fsname=<fsname>\
    --failover=<failover mds hostdesc> (--clumanager|--heartbeat)\
    <mds partition>
mount.lustre <mds partition> <mds mtpt>
```

oss

```
mkfs.lustre --ost --fsname=<fsname>\
    --mgtnode=<mds hostdesc>[,<failover mds hostdesc>]\
    --failover=<failover oss hostdesc> (--clumanager|--heartbeat)\
    <oss partition>
mount.lustre <oss partition> <oss mtpt>
```

client

```
mount.lustre <mds hostdesc>[,<failover mds hostdesc>]:/<fsname>\
    <cli mtpt>
```

6.5 Stopping a service

On the OSS or MDS in question

mds/oss

```
lfs export stop <mds|oss mtpt>
umount.lustre <mds|oss mtpt>
```

This stops the server unconditionally. It removes client export information and does not perform a failover to another server. Issuing the umount command without stopping the service will report that the file system is busy.

6.6 Forcing failover

On the OSS or MDS one wishes to failover to the failover host issue the following command:

mds/oss

```
umount -f <mds|oss mtpt>
```

This perserves client export information. When the service for the target is restarted on the failover node, or on the same node, the clients will reconnect to this server.

6.7 Re-addressing a failover node

Readdressing the failover node is an example of a dynamic update. If such updates are of a permanent nature, clients only derive value from this if they have a config cache file, see??.

```
lfs export --setfailover=<host desc> <fsname> \
<mds|oss partition label>
```

This command is run on the node serving the target provided by the partition. It replaces the currently recorded failover host descriptor, if any, and writes the one given on the command line. This command requests a modification at the management node.

Alternatively

```
lfs export --host=<host desc> --setfailover=<hostdesc> --label=<target label>  <lustre
```

can be issued on any server.

6.8 Local mounts through network

The following configuration sets up MDS, OSS on one node. The MDS will mount a client file system using the loopback portals transport, which has host-descriptor **lo**. The OSS mounts its file system as a loopback mount of that of the MDS through the **lb** option to mount.lustre. There is a further client file system connecting to this through TCP networking, and finally a client file system using loopback portals transport:

mds/oss/client

```
mkfs.lustre --fsname=<fsname> --mdt <mds partition>
mkfs.lustre --fsname=<fsname> --ost --mgtnode=lo  <oss partition>
mount.lustre <mds partition> <mds mtpt>
mount.lustre -o loop <oss partition> <oss mtpt>
mount.lustre localhost:<fsname> <cli mtpt>
mount.lustre lo:<fsname> <cli mtpt>
```

6.9 Start Lustre on a client or server node, ignoring the management network

Add the following option to the mount command, where eth0 is the interface that should not be used as a transport:

```
mount.lustre -o ignoreif=eth0
```

Without this option, all available addresses on all interfaces will be bonded.

6.10 Mounting a Lustre server but no service

Add the following mount option to the server mount commands:

```
mount.lustre -o nosvc
```

Note: OSS's will still be able to connect to integrate into the file system

6.11 Adding an OSS/MDS to an existing file system

On the new OSS or MDS:

mds/oss:

```
mkfs.lustre --fsname=<fsname> (--ost|--mdt) --mgtnode=<mds hostdesc>[,<other mds hostde
mount.lustre <partition> <new mtpt>
```

To add an OSS or MDS target at a particular index in the LOV or LMV add the –**stripeindex**=<**index**> option to the mkfs.lustre command.

6.12 Security policies on some interface for the MDS

! I think this needs to be in the network section !

The following commands configure the service for <fsname> to have weak security on elan, and for all traffic originating from a class C family of IP 192.168.1.0/0.0.0.255:

mds/oss:

```
mkfs.lustre --fsname=<fsname> -secure=elan -secure=192.168.1.0/0.0.0.255 <mds partition
```

6.13 Configure a client with a persistent write back cache

On the client format a loop device cache for oss and mds cache combined with mkfs.lustre:

```
mkfs.lustre --loop --ost --mdt --cachefor=<mds hostdesc>[,<mds2 hostdesc> ..]:/<fsname>
```

Now simply mount the cache:

```
mount.lustre -o loop <cache file> <cli mtpt>
```

6.14 Configure a replicating proxy cluster

Select MDS and OSS nodes acting as a proxy cluster

```
mkfs.lustre -mds -proxyfor=<mds hostdesc>[,<mds2 hostdesc>,...] <proxy device>
mkfs.lustre -oss mds=<mds proxy hostdesc> <proxy oss1 device>
mkfs.lustre -oss mds=<mds proxy hostdesc> <proxy oss2 device>
```

It is possible to specify only an MDS proxy or only an OSS proxy. To connect a client to the file system through a proxy

```
mount.lustre <mds proxy hostdesc>//fsname
```

6.15 Pools

To define a pool of OSS's for a Lustre file system, each with a stripe and access pattern, issue the following mkfs.lustre command:

```
mkfs.lustre --mdt --ostpool=<pool1name> --stripecount=<cnt1> --access=c@1,r@2,w@3 --ost
```

Here network 1 has create (i.e. read, write and create access), network 2 only read-only access and network 3 has read-write access, but cannot create objects. To add a new pool, on a running or non-mounted Lustre server

```
lfs export --persist --ostpool=<pool3name> ..... (<device>|<label>|<mds mtpt>)
```

To add an OSS into a certain pool configure them as:

```
mkfs.lustre -oss -pool=<poolname> -mds=<mds hostdesc> -fsname=<fsname> <oss partition>
```

6.16 Target configuration

Often it is desirable to protect subtrees of a file system quite differently.

```
mkfs.lustre -mds -fsname=<fsname> <mds device>
mount.lustre -o nosvc
lfs export fsname=<fsname> [root_squash,no_setuid,net-desc:perm]
lfs export fsname=<fsname>/<subdir> [root_squash] [ro] [nosetuid]  [net-desc:perm]
```

6.17 Echo client & Single OST server

```
mkfs.lustre --mds --oss -echo <mds oss device>
mount.lustre <mds oss device> /mnt/lustre
insmod echoclient mds=<mds oss hostdesc> lustre.o
```

Now this client can be driven lctl or through /proc. Or if the lov on the echo client is not desirable, use

```
insmod echoclient mds=<mds oss hostdesc> lov=no lustre.o
```

Without this option the echo client is layered, as usual on the lov which layers on the osc. With the lov=no option, the echo client speaks directly to the OSC. This option can also be used with llite.

6.18 Striped OST echo server

```
mkfs.lustre -mds <mds device>
mount.lustre <mds device> <mds mtpt>
insmod echoserver mds=<mds hostdesc> lustre.o
```

The latter command is issued on each OSS in the stripes and it adds the OST to the MDS LOV desscriptor.

6.19 Long term failure and removal of OSS targets

An OSS target can be operating normally. In this mode, if a client does not get a response it will wait indefinitely for the OSS target or its failover target to be present and process the request. The wait is interruptible, should a client want to abort waits.

```
lfs export fail osshost=<oss hostdesc> osspart=<oss partition> <fs mtpt>
```

After this command is issued the OSC's will update the information about this target and they will fail all writes, by returning errors on the client. This command can be issued on any node, provided administrative priviliges are available. All I/O read and write to the OST fails in this mode.

A next decision step regarding the OST is that it may never return to the cluster. Now it needs to be administratively removed. Some customers prefer to have no further complaints from the removed OST and return 0's on reads and make writes successful by writing new objects. To put the OST in that mode:

```
lfs export remove osshost=<oss hostdesc> osspart=<oss partition> <fs mtpt>
```

6.20 Migration

```
mkfs.lustre -oss -mgtnode=<mds hostspec> -fsname=<fsname> <new oss device>
mount.lustre -nosvc <new oss device> <oss mtpt>
lfs migrate -srcossname=<oss hostdesc> -srcosspart=<oss partition> -dstossname=<oss hos
```

Similarly a new pool of OSS's may be added and data from one pool can be migrated to another:

```
    lfs migrate -srcpool=<pool name> -dstpool=<pool name> <fs mtpt>
```

MDS migration is arranged similarly.

7 State management

To facilitate atomic updates, the configuration state is written transactionally in shared storage, accessible by any server and its failover partner. The configuration consists of several parts:

1. A version number
2. Change entries from all previous versions, tagged to allow any version to roll forward. Change entries exist for adding, failing, removing and readdressing servers.
3. A full record of the last version:
   (a) A number of client parameters can be fetched, such as timeouts.
   (b) All nodes will fetch the latest LOV descriptors which contain the pool names, stripe policies, permissions and OSS nodes for the pool. LOV descriptors can be very large - approximately 200 OSS's can be stored per 4K page. The OSS node descriptor contains a host descriptor and alternate host descriptor and an index.
   (c) An LMV descriptor, similar to the LOV descriptor, but typically smaller.
   (d) The routing table.

In addition to MDS managed state, it is assumed that the site can manage distribution of mount maps and /etc/modules.conf to manage parameters that cannot be fetched from servers (e.g. router nodes).

8 Implementation roadmap

8.1 Current state

- libcfs and portals are still separate modules, but the network can be initialised fully (as described above) with `insmod portals` and all required NAL modules are loaded on demand.
- lconf has been altered to edit out all network configuration.

- portals router is included in the portals module. It is consulted when first sending into the network, but NALs still interface directly to it for message forwarding.
- Socknal and qswnal implement all tunables as module variables. Other NALs will be converted as required.
- Socknal can discover all local interfaces. The same old interface matching code is still there (i.e. a client with 1 interface connecting to a server with 2 interfaces sticks to the "published" interface of the server. We'll either have to implement a "reconnect on this interface" protocol, or get convinced that clients connecting to all server interfaces is a good idea to achieve load balance in multi-NIC server with single-NIC client situations.
- `lctl network` prints all local NIDs.
- `lctl network unconfigure` is required at teardown so that portals can release the NALs and allow them to unload. Once all the NALs have been unloaded, portals can be unloaded. lconf has been altered to run this and attempt to unload all known NALs when it unloads the ptlrpc module.
- libcfs/nidstring.c implements common NID/string operations. It can be included in the kernel, from liblustre and from general userspace utilities.
- lustre configuration still uses XML. The NIDs specified in this XML must be "new style" address@network NIDs.
- lustre only uses a single portals NI.
- All this is on portals b_hd_newconfig and lustre b1_4_newconfig.

8.2 Next steps

8.2.1 Landing on portals HEAD and lustre HEAD/b1_4/b_cray

Imminent

8.2.2 Connection handling

Ptlrpc_connection structures are removed. Client obds have two slots for addresses, primary and failover (no multiple failover alternatives at this point). During failover the role of primary and secondary is switched.

XML would still be used to specify the address and failover address of OST's and MDT's, for storage in the confobd llog. No lctl net commands will be needed anymore at this point. Portals should be branched and the branch should be be committed and used with Lustre HEAD (no routing on HEAD).

8.2.3 Packet flagging

insmod can take arguments such that the socknal flags events with an integer, when packets coming in from one or more ranges of IP addresses.

insmod can take an argument to declare that all packets on a certain interface or NAL are flagged with an integer.

Each of these special packet sources must optionally be able to use a different integer to flag events (so that for example we can have different flags for *no-security* and *no-encryption.*

Commit code for Lustre HEAD

8.3 Portals - phase 2

Subject to funding.

8.3.1 Configuration cache files

8.3.2 Dynamic updates for router failure notifications

9 Outstanding issues

1. It seems that non-routed multiple network installations may benefit from the nettype. Produce examples
2. Consider the role of the Lustre filesets and mount objects.
3. Consider raid 1 and 5 repair commands for OST's
4. Consider if implementing the target implementation at once is possible
5. Index value

10 Changelog

2005/03/15 First draft

2005/03/16 Add more variety to targets

2005/03/22 Describe Lustre.o, give a very involved networking example. Try to clarify management tree. Become clear that insmod parameters configure networking and mount parameters configure file systems. Become clear that without management nodes performing online updates of configurations this is equally useful as current configuration mechanisms.

2005/03/22 Explain how fsnames need to be combined with acronyms like **oss** and an integer **index** to allow everything to be crammed into 16 character ext3 labels.

2005/04/02 Incorporate internal feedback. Incorporate comments from LLNL.

1. Clarify the role of NIDs and host interface addresses in "Usage of the portals api"
2. Objections to the ignoreif parameter, we like what we have, perhaps use an option *only-configured-ifs*
3. I have come to like the pools idea to address uniform allocation policies. Removing the security bits from it as Terry proposes is perhaps good.
4. Include description of failure handling during startup and target addition.
5. Describe update protocol in more detail
6. Discuss the umount.lustre interaction with failover services.
7. Give an example where stripe index is used. Can we do this automatically? Doesn't seem easy.

2005/04/03 Incorporate comments from Evan Felix: be more clear about targets and servers.

2005/05/23 Network configuration brought up to date.

2005/05/28 Screen the document. Questions remain about:

1. Easy handling of target id names
2. prepare for MDT pools, so change –pool to –ostpool

2005/05/31

1. Remove flagging. Instead we add an IP source address to the event. This separates Lustre security policy (which can be per file system) from flagging policy.
2. Note that data migration has to respect pools.