NFS/RDMA Linux Client

Tom Talpey
Network Appliance, Inc.
tmt@netapp.com
Outline

• NFS/RDMA Protocol(s)
• Implementation on Linux
• Results
• Next steps
What is NFS/RDMA

• A binding of NFS v2, v3, v4 atop RDMA transport such as Infiniband, iWARP
• A significant performance optimization
• An enabler for NAS in the high-end
Benefits of RDMA

- Reduced Client Overhead
- Data copy avoidance (zero-copy)
- Userspace I/O (OS Bypass)
- Reduced latency
- Increased throughput, ops/sec
Followon NFS/RDMA Benefits

• Protocol enhancements and extensions
  – Databases, cluster computing, etc
• Scalable cluster/distributed filesystem
• As we raise the “NAS bar”, the protocol should express richer semantics
What has been proposed

• IETF NFSv4 Working Group
• From the bottom up:
  – RPC/RDMA
  – NFS RDMA binding
  – NFSv4 Transport enhancements
    • Sessions
    • Exactly-once semantics
RPC/RDMA

• Core RDMA transport binding for RPC in general
• Provides
  – Encoding, etc
  – Inline and Direct (RDMA chunk) transfer
  – Credits
• http://www.ietf.org/internet-drafts/draft-callaghan-rpcrdma-00.txt
NFS Direct

• NFS binding for RPC/RDMA
• Provides
  – Inline and Direct (RDMA) NFS RPC definitions
  – “What gets chunked”
• http://www.ietf.org/internet-drafts/draft-callaghan-nfsdirect-00.txt
NFSv4 RDMA and Sessions

• Transport Enhancement for NFSv4
• Provides
  – Session concept
  – Exactly-once semantics
  – General for TCP and RDMA
• http://www.ietf.org/internet-drafts/draft-talpey-nfsv4-rdma-sess-01.txt
NFS RDMA Problem Statement

- IETF Problem Statement for NFS over RDMA
- Provides
  - Rationale
  - Outlines requirements
  - IETF-chartered first step
NFS RDMA Requirements

• IETF Requirements doc for NFS over RDMA
  • Provides
    – Detailed requirements
    – Input to RDDP group
    – IETF-chartered first step
• http://www.ietf.org/internet-drafts/draft-callaghan-nfsrdmareq-00.txt
The Documents Together:

• Form the basis for a complete NFS over RDMA solution
• All NFS versions, and general RPC
• Do not fundamentally propose new NFS features (but enable a few)
Applying to NFSv3

- Immediate performance benefit
- Straightforward integration with existing implementation
- High market acceptance
- “NFS on Steroids”
- Side protocols (NLM) problematic
Applying to NFSv4+

• Performance
• Enhanced correctness
  – “The goodness of NFSv4”
  – Exactly-once semantics (“EOS”)
  – No side protocols / side connections
• Sessions
  – Trunking
  – Failover
  – Efficient resource management
  – (Other benefits from EOS)
  – For both TCP and RDMA
Roadmap

• Early win: NFSv3 on IB
• Prepare the Transport: NFSv4 Sessions
• Enable the applications by extending the protocol
• Employ (and foster) iWARP
• NFSv4/RDMA as cluster FS
Client Implementation Goals

• Support NFS/RDMA
• Support other transports:
  – TOE
  – IPv6
  – “Bypass” (pNFS)
• Integrate with Linux
Existing Linux RPC support

- Single module – sunrpc.o
- Only IPPROTO_{TCP,UDP}
- Only kernel sockets API
- Much specific knowledge roto-tilled:
  - Stream/dgram (framing needed)
  - Connection oriented (reconnect needed)
  - Reliable (retransmit needed)
- Endpoint is 1-1 per xprt (mount)
Solution: RPC Transport Switch

- Abstraction for transport type
- One each for
  - TCP
  - UDP
  - RDMA
  - More to come
NFS-RDMA
Client Software Stack

NFS version 3
(Unmodified Linux VFS)

Linux RPC
Transport Switch

RPC-TCP
RPC-UDP
RPC-RDMA

Kernel sockets

Ethernet

iWARP
IB

...
Transport Switch Vector

New pointer in the “struct rpc_xprt”:

/* abstract functions provided by a transport */
struct rpc_xprt_procs {
    void * (*allocate)(struct rpc_xprt *, struct rpc_task *, unsigned int);
    int       (*sendmsg)(struct rpc_xprt *, struct rpc_rqst *);
    void     (*free)(struct rpc_xprt *, struct rpc_task *, void *);
    void     (*reconnect)(struct rpc_task *);
    int       (*create)(struct rpc_xprt *, struct xprt_create_data *);
    int       (*destroy)(struct rpc_xprt *);
    void     (*close)(struct rpc_xprt *);
};
Socket Transport Creation

#define RPC_MAX_TRANSPORTS 16
#define RPC_XPRT_TCP 0 /* sock_create_data */
#define RPC_XPRT_UDP 1 /* sock_create_data */
#define RPC_XPRT_RDMA 2 /* rdma_create_data */

struct sock_create_data {
    struct sockaddr_in srvaddr;
    struct rpc_timeout * timeo;
};
RDMA Transport Creation

struct rdma_create_data {
    /* Generic fields */
    struct sockaddr_in    srvaddr;
    struct rpc_timeout *  timeo;

    /* Server RDMA address and port */
    struct sockaddr      addr;
    u64                  port;

    /* Per-mount tuning */
    int                  max_requests; /* max credits/requests in flight */
    int                  rsize;        /* server r/w sizes (mount opts) */
    int                  wsize;

    /* Per-server configuration - must be <= remote settings */
    int                  max_inline_send; /* Inline data max */
    int                  max_inline_recv; /* Inline data max */
    int                  padding;       /* Inline write pad */
};
Transport Switch Registry

/*
 * rpc_transport represents a transport for use by RPC.
 * This is provided by each transport.
 */

struct rpc_transport {
    char name[8];
    int transport_number;
    struct rpc_xprt_procs procs;
};

int xprt_register(struct rpc_transport *);
int xprt_unregister(struct rpc_transport *);

/* Alternative for xprt_create_proto that is transport-switch aware. */
struct rpc_xprt *xprt_create_transport(struct xprt_create_data *);
Transport Hooks

• Each transport registers with switch
• NFS mount (and others) specify transport type and per-transport create data
• Transport gets control via xprt_procs
• Can unregister/unload
Lifecycle of an RPC

1. NFS VFS processing
2. RPC Transport allocation
3. NFS marshalling
4. RPC sendmsg (Includes transport header marshalling)
5. Server processing
6. RPC Demux (soft ISR)
7. NFS completion
8. RPC Free

New switch abstractions
Memory Representation

- Leverage Linux implementation heavily
- Use allocation hook to set up preregistered request/reply buffers (headers)
- Use iovec (<= 2.4.19) or pagelist (>= 2.4.20) to map any data
Memory Representation

<=2.4.19:

IOV (header) → IOV (data) → ...

>=2.4.20:

IOV (header) → Page (data) → Page (data) → IOV (tail)

- Header segment always copied to inline
  - All metadata ops, small reads/writes “pulled up”
- Data segments translated directly to rpcrdma “chunks”
- No need for NFS layer to become involved
Transfer models

- Follow the RPC/RDMA protocol
- Full inline (no chunking)
- Direct read, write (via write/read chunks, respectively)
- “Overflow transfer” via reply chunks or position-0 requests
- Write padding supported
Inline I/O Operations

• “Small ops”: metadata and inline Read and Write
  – Just like regular RPC
  – Pre-allocated buffers, pre-registered with the transport
  – Configurable message size limit
  – Low transport latency, simple model
  – Header padding for write data alignment
Direct I/O Operations

- Direct Read and Write
  - 3-part transfer
    - Server initiates RDMA operation
    - Buffer placed per request
    - Used for large messages
    - Zero-copy, low CPU cost
Overflow Direct Operations

- Large metadata transfers
  - 3-part transfer
  - Client expresses entire request or reply as chunk
  - Server performs RDMA operation
  - Used when request or response size > max
    - e.g. rename, readlink, readdir
  - Provides correctness for corner cases
    - Not on read/write path
Buffer Cache

• Operation to Linux buffer cache fully supported
• RDMA to/from cache, bcopy to/from user
• Improved overhead from sockets case
  – Protocol offload, copy avoidance
• Convenient because buffer cache is in kernel address space
Direct I/O

- User directio fully supported in appropriate kernels (>= 2.4.19)
- User pages passed as pagelist by NFS
- Pages are registered for RDMA
- Zero-copy, zero-touch
- When physical addressing in use, no kmap/kunmap is required (no TLB inval)
Client Implementation

- Patch for sunrpc (transport switch)
- RPC/RDMA module
  - 3000 lines of code, 2 headers, 3 C files
- kDAPL “null” provider
- IB kDAPL providers under way
Client Implementation

• Available as open source
  – BSD-style license
  – www.sourceforge.net/projects/nfs-rdma

• Supported Linuxes:
  – RedHat 7.3 (2.4.18)
  – SuSE 8 Enterprise (2.4.19)
  – RHEL 3.0 (2.4.21)
  – 2.6 support under way
kDAPL

• Kernel Direct Access Programming Library
• Transport API for RDMA
  – Implemented as part of each driver, with global registry
• Supports iWARP, Infiniband, VIA
• Open reference implementation
• www.datcollaborative.org
• www.sourceforge.net/projects/dapl
Performance

1. Streaming throughput
2. Transactional throughput
3. Seat-of-pants
   • Tests run on Dell 2650
     – SuSE Linux Enterprise Server 8 (~2.4.19)
     – 4x Infiniband connection (10Gb)
     – 2.4GHz dual Xeon
     – Hyperthreading disabled
     – NetApp 960 Filer(s)
Streaming Throughput

- 4K synchronous random reads from server cache
  - i.e. single thread, no caching, no readahead.
- Achieves ~350MBytes/sec
  - This includes one data copy from kernel->user!
- Uses only 20% of client CPU
- RDMA, low latency, protocol offload all contribute
Transactional Throughput

- OLTP benchmark (4-way CPU)
- Compared to 1Gb NFS/TCP, 2Gb Fibre Channel
  - These runs are *not* bandwidth limited
- NFS runs encounter 1 data copy (database !O_DIRECT)

<table>
<thead>
<tr>
<th></th>
<th>OLTP ops</th>
<th>System time</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS/TCP</td>
<td>17K</td>
<td>21%</td>
<td>Idle time</td>
</tr>
<tr>
<td>Fibre</td>
<td>21K</td>
<td>20%</td>
<td>Host CPU</td>
</tr>
<tr>
<td>RDMA</td>
<td>20K</td>
<td>26%</td>
<td>Host CPU (data copy)</td>
</tr>
</tbody>
</table>
Seat-of-pants

- Build the Linux kernel
- NFS runs encounter significant creat/open/close attribute traffic – expect much better w/v4

<table>
<thead>
<tr>
<th></th>
<th>Build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local disk</td>
<td>3:05</td>
</tr>
<tr>
<td>NFS/TCP</td>
<td>6:10</td>
</tr>
<tr>
<td>RDMA</td>
<td>4:10</td>
</tr>
</tbody>
</table>
Next Steps

• Transport switch
  – Clean up, generalize
  – Integrate with 2.6.x
  – Expose transport creation args via mount
Next Steps

• Linux Infiniband support
• For base kernel, also in distributions
  – Infiniband vendors
• With kDAPL support
Next Steps

- NFSv4/RDMA/Sessions
- UMich CITI
- http://www.citi.umich.edu/projects/rdma/
Next Steps

- NFS/RDMA Linux Server
- (TBD)
Next Steps

• Other applications of transport switch
  – TOE
    • Non kernel-sockets TOE API may add efficiency
  – IPv6
    • Better express addressing, transport differences
  – pNFS (parallel NFS)
    • Fibre Channel / iSCSI “bypass”
  – Multiple TCP endpoints
    • Simple trunked/failover mountpoints
Next Steps

- iWARP support
- Emerging technology in 2004
Backup – NFSv4/Sessions
The Proposal

• Add a session to NFSv4
• Enable operation on single connection
  – Firewall-friendly
• Enable multiple connections for trunking, multipathing
• Enable RDMA accounting (credits, etc)
• *Provide Exactly-Once semantics*
• Transport-independent
5 new ops

• SESSION_CREATE
• SESSION_BIND
• SESSION_DESTROY
• OPERATION_CONTROL
• CB_CREDITRECALL
Channels versus Connections

• Channel: a connection bound to a specific purpose:
  – Operations (1 or more connections)
  – Callbacks (typically 1 connection)
• Multiple connections per client, multiple channels per connection
  – Many-to-many relationship
• All operations require a streamid/channelid
  – Encoded into COMPOUND
Session Connection Model

- Client connects to server
- First time only:
  - New session via SESSION_CREATE
- Initialize channel:
  - Bind “channel” via SESSION_BIND
  - May bind operations, callback to same connection
  - May connect additional times
    - Trunking, multipathing, failover, etc.
- CCM fits perfectly here
- If connection lost, may reconnect to existing session
- When done:
  - Destroy session context via SESSION_DESTROY
Example Session – single connection
Example Session – multiple connections
Example Session – single connection

• Resource-friendly
• Firewall-friendly
• No performance impact
• Isn’t this the way callbacks should have been spec’ed?
Exactly-Once Semantics

- Highly desirable, but never achievable
- Need flow control (N), operation sizing (M) in order to support RDMA
- Flow control provides an “ack window”
  - Use this to retire response cache entries
- $N \times M = \text{response cache size}$
- Session provides accounting and storage
- Done!
Streamid

• A per-operation identifier in the range 0..N-1 of server’s current flow control  
  – In effect, an index into an array of legal in-progress ops

• Highly efficient processing – no lookup

• Used in conjunction with RPC transaction id to maintain duplicate request cache
Chaining

- Problem: COMPOUND restricted in length at session negotiation
- Chaining provides strict sequencing of requests
  - “compound for compounds”
- Start, middle, end flags (and none)
- Maintains current and saved filehandles like COMPOUND
Connection model and negotiation

- Simplest form – no session at all
- Session binding enables use of RDMA
  - Per-channel (connection) RDMA mode
  - Mix TCP and RDMA channels per-client!
- TCP mode if either RDMA mode is off
- Dynamic enabling of RDMA at session binding
  - After RDMA mode, sizes, credits, etc exchanged
- Statically enabled RDMA (e.g. Infiniband) also supported
  - Requires preposted buffer
V4 Protocol integration

- Piggyback on existing COMPOUND
- New OPERATION_CONTROL first in each session COMPOUND request and reply
- Conveys channelid, streamid, and chaining
V4 efficiencies

• No need for sequenceid
  – Field will stay, but ignored under a session
• No need for clientid per-op
  – Clientid may be provided as zero
• Each request within session renews leases
• OPEN_CONFIRM not needed
• CCM is enabled