Running SMB3.x over RDMA on Linux platforms

Mark Rabinovich
Visuality Systems

John Kim
Mellanox Technologies
Prehistory

- NQ CIFS is an implementation of SMB client/server for the embedded world:
  - Consumer devices: printers, MFP, routers, STB, etc.
  - Mobile devices: smart phones, tablets
  - Industrial Automation, Medica, Aerospace and Defense
  - Anything else that is neither PC nor MAC nor Samba..
- The last version – NQE – supports basic SMB3.0, mostly towards data encryption.
- SMB over RDMA is only supported on Windows platforms
- Currently we are developing a storage-level version of SMB Server. The presentation is about this venture.
Plans and Goals

- Enterprise-level performance with low latencies
- 10,000 connections at least
- Pluggable transports
- Pluggable VFS
We will mostly talk about the SMBD transport
Architecture remarks

- Socket-based transport uses `epoll()` or alike.
- A **pluggable transport** may be:
  - Fast, zero-copy sockets
  - Proprietary transport protocol
  - Etc.
- **Pluggable transport** conforms to Transport API.
- **Core** translates SMBs into NTFS semantics.
- **Embedded VFS** converts NTFS semantics into POSIX semantics (or whatever the local FS is).
- **Pluggable VFS** translates NTFS semantics into Storage primitives.
- **Pluggable VFS** conforms to VFS API.

*The SMBD transport is of a particular interest for this session.*
Dataflow in layers

- SMB
- Transport API
- SMBD
- RDMA API
- RDMA

- An example of involved layers as applicable to the subject of this session
Dataflow explained

- Abstract APIs allows replaceable modules
- SMB Direct is one of the possible transports
- “RDMA” actually means two services:
  - Packet delivery
  - Payload delivery, which is applicable to Read(s)/Write(s) and assumes RDMA
- Our RDMA layer is also replaceable
SMBD – a drill-down view

![Diagram of SMBD components]

- SMB engine
- SMBD
  - Packet service
  - RDMA service
    - allocate
    - RDMA
- Connection listener
- Event dispatcher
- Buffer management
- Release callbacks

Incoming SMB request
Send
Read/write
**SMBD remarks**

- Two thread run:
  - New connection listener registers a callback with RDMA API. On callback:
    - Handles connection request.
    - Handles connection acknowledgement. Submits several request buffers to RDMA receive queue.
    - Handles disconnect
  - Data event listener registers a callback with RDMA API. On callback:
    - On Send complete releases the buffer
    - On Receive complete accepts the incoming packet and delegates it to SMB. Sometimes assembles an SMB.
    - On Read complete and Write complete releases a 1MB buffer.
Buffer sizes

Each protocol negotiates its own buffer sizes

- **SMB**
  - Max Transaction Size
  - Max Read Size
  - Max Write Size

- **SMBD**
  - Preferred Send Size
  - Max Read/Write Size
  - Max Receive Size
  - Max Fragmented Size
Buffer sizes (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Windows 2012</th>
<th>NQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Transaction Size</td>
<td>1048576</td>
<td>1048576</td>
</tr>
<tr>
<td>Max Read Size</td>
<td>1048576</td>
<td>1048576</td>
</tr>
<tr>
<td>Max Write Size</td>
<td>1048576</td>
<td>1048576</td>
</tr>
<tr>
<td>SMBD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preferred Send Size</td>
<td>1364</td>
<td>8192</td>
</tr>
<tr>
<td>Max Read/Write Size</td>
<td>1048576</td>
<td>1048576</td>
</tr>
<tr>
<td>Max Receive Size</td>
<td>8192</td>
<td>8192</td>
</tr>
<tr>
<td>Max Fragmented Size</td>
<td>1048576</td>
<td>1048576</td>
</tr>
</tbody>
</table>
## Buffer vs. iovec

<table>
<thead>
<tr>
<th>IOVEC</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grants zero-copy i/o</td>
<td>Proper buffer management (see below) will not also require data copy</td>
</tr>
<tr>
<td>Requires more complicated code and also causes (apparently -minor) performance decrease</td>
<td>SMBD over RDMA grants packet delivery in one piece if it fits into the receive buffer</td>
</tr>
<tr>
<td></td>
<td>Most SMBs fit into one buffer and are delivered in one piece</td>
</tr>
</tbody>
</table>

**Conclusions**

- Majority of SMBs will fit into one receive buffer
- SMBD assembly is rarely needed
- Buffers are sufficient – no need for iovec
Buffer management

- “Small” buffer is 8192 bytes long (Max Receive Size!!)
  - Most SMB requests (except, maybe, for some odd IOCTLs)
  - Most SMB responses (except for Query Directory)
  - Reads and Writes normally small since RDMA is enforced. Some exceptions happen - see below
  - Almost all receive buffers are “small”
- “Medium” buffer is 64K bytes long
  - Query Directory responses
  - Some RPC responses (e.g., NetrEnumShares)
  - Odd requests/response (Query Quota, IOCTL)
- “Big” buffer is 1M bytes long
  - RDMA reads and writes as SMB Read or Write payload
  - Non-RDMA Reads and Writes (rarely happens)
SMBD Fragmentation and Assembly

- Fragmentation may happen on transmitting a long response (e.g., `Query Directory`). Fragmentation is achieved by using the same buffer and sending its contexts with shifting offsets.
  - This is available due to the RDMA layer’s capability of submitting the same buffer with different offsets.
  - Fragmentation does not cause data copy and applies virtually zero overhead.
- Assembly may happen upon receiving a long request.
  - This almost never happens since the vast majority of requests are small, including RDMA reads and writes.
  - On a (rare) message assembly, data copy happens.
  - A long request is assembled in a “medium” or “big” buffer.
Buffer pre-allocation

- On a 10Gbs network memory allocation may become a bottleneck.
- NQ uses an abstract API for memory (pre)allocation which counts on the underlying memory management mechanism.
- This is not enough so NQ pre-allocates buffers in:
  - SMB Direct layer
  - SMB layer
SMBD Crediting

- SMBD requests and SMBD responses imply a simple crediting algorithm.
- Credits == number of receive buffers per a connection.
RDMA Remarks

- Listens to connection events using `rdma_get_cm_event()`.  
  - New connection request  
  - New connection acknowledge  
  - Disconnect
- Listens to data events using `ibv_get_cq_event()` and `ibv_poll_cq()`.  
  - An incoming message placed into a receive buffer  
  - An outgoing packet was sent  
  - RDMA transfer completed

Any of the above causes a callback to the SMBD layer.
RDMA Remarks (cont.)

- Queues a buffer for receiving through `rdma_post_recv()`. Receive completion on this buffer will generate an event.
- Queues a buffer for sending through `rdma_post_send()`. Send completion will generate an event.
- Queues a buffer for RDMA read/write through `rdma_post_read()`. A completion on this buffer will generate an event.
Test Environment

- The Windows client side:
  - HP Proliant ML350P Server
  - Runs Windows 2012
  - Mellanox ConnectX-3 network adapter

- The Linux server side:
  - HP Proliant ML350P Server
  - Runs Redhat 2.6.32
  - Runs NQ CIFS Server 8.0 (prototype)
  - Mellanox ConnectX-3 network adapter

- The Mellanox card was tested in both InfiniBand mode and Ethernet mode.
The Testbench

- The aim was to validate SMB over RMDA
- Currently out of scope:
  - Multiple connections
  - Random operations
  - Small chunk operations
- We were only uploading and downloading a 10GB file.
- We plugged Embedded VFS with POSIX backend. On 10Gbs this is definitely a bottleneck.
- Then we measured with a dummy backend.
Performance

Embedded VFS with POSIX backend.

Embedded VFS with a dummy backend.
Numbers

- We used Embedded VFS with a dummy backend.
- The network is InfiniBand
- Download: 940 MByte/sec
- Upload: 1,400 MByte/sec
- These results are similar to Win-Win numbers
Thank you