Implementing SMB Direct for Linux SMB Client

Long Li
Microsoft
Agenda

• Introduce to SMB Direct and RDMA
• Implement SMB Direct
  • RDMA send/receive
  • RDMA read/write through memory registration
• Profiling and optimization
• Some benchmark data
Linux SMB kernel Client

CIFS.KO

/lib/moduels/`uname –r`/kernel/fs/cifs/cifs.ko

mount –t cifs
SMB Direct

- Transferring SMB packets over RDMA
  - Infiniband
  - RoCE (RDMA over Converged Ethernet)
  - iWARP (IETF RDMA over TCP)
- Introduced in SMB 3.0 with Windows 2012

<table>
<thead>
<tr>
<th>Windows Server 2012</th>
<th>SMB 3.0</th>
<th>New features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Server 2012 R2</td>
<td>SMB 3.02</td>
<td>Remote invalidation</td>
</tr>
<tr>
<td>Windows Server 2016</td>
<td>SMB 3.1.1</td>
<td></td>
</tr>
</tbody>
</table>
RDMA programming interface

• In kernel programming interface
  • drivers/infiniband/core
  • Verbs: ib_core
  • Connection Management: rdma_cm

• Works on all RDMA transport
  • Infiniband
  • RoCE
  • iWARP

• Avoid using verbs that are specific to Infiniband
SMB Direct as a transport

User mode

I/O workload

Kernel mode

VFS

Page cache

SMB2

TCP socket

CIFS.KO

VFS

Page cache

SMB2

SMB Direct
Transfer data with SMB Direct

• RDMA send and receive
  • Similar to TCP socket interface, but with no data copy in most cases

• RDMA read and write
  • Overlap local CPU and communication
  • Reduce CPU overhead on send sider

• Connection based
  • One RC (Reliable Connection) Queue Pair per connection
  • Completion Queue is used to signaling I/O complete
RDMA Send/Recv vs TCP

• Stream vs datagram
• Both have the option of establishing reliable connection
  • SMB Direct connection is a reliable connection
  • Port 445 for Infiniband and RoCE environments
  • Port 5445 for iWARP environments
• TCP works with stream
  • Send whatever you want and receive the way you want it
• RDMA works with datagram
  • Application figures out how to send them
  • May need to do segmentation on send
  • Reassemble data payloads on receive
RDMA Send/Recv vs TCP

• Memory management
• TCP socket
  • TCP maintains send and receive buffers and communicate with peer on flow control
• RDMA
  • Application manages its own buffers
    • Send -> no receive?
  • Application needs to do flow control
    • SMB Direct uses a credit system
    • No send-credits? Can’t send data.
RDMA Send/Recv

SMB Client

SMB2

Wait for credits

Data

SMB Direct

Data

Data

Data

SMB Server

SMB2

Data

SMB Direct

Data

Data

Data

Reassemble
RDMA Send/Recv

- CPU is doing all the hard work of packet segmentation and reassembly
- Not the best way to send or receive a large packet
  - Slower than most TCP hardware
  - Today most of TCP based NIC support hardware offloading
- SMB Direct uses RDMA send/recv for smaller packets
  - Default for packet size less than 4k bytes
SMB Direct credit system

SMB2

SMB Direct

Wait for credits

Number of buffers are limited

Reassemble

Data

Data

Data
SMB Direct credit system

• Send credits
  • Decreased on each RDMA send
  • Receiving peer guarantees a RDMA recv buffer is posted for this send

• Credits are requested and granted in SMB Direct packet
SMB Direct credit system

• Running out of credits?
  • Some SMB commands send or receive lots of packet
  • One side keeps sending to the other side, no response is needed
  • Eventually the send runs out of send credits

• SMB Direct packet without payload
  • Extend credits to peer
  • Keep transport flowing
  • Should send as soon as new buffers are make available to post receive
RDMA Send/Recv

• How about large packets for file I/O?
• Typically SMB negotiates with I/O size as large as 1MB
RDMA Read/Write

SMB Client

SMB Server

SMB2

Data

Wait for credits

SMB Direct

Data

Data

Data
RDMA Read/Write

SMB Client > SMB Server
Transfer I/O via Server initiated RDMA read

SMB2 Data
Wait for credits

SMB Direct Data
SMB Direct packet describing the memory location in SMB Client
Memory registration

• Client needs to tell Server where to write or read the data from its memory
• Memory is registered for RDMA
  • May not always be mapped to virtual address
  • I/O data are described as pages
• Correct permission is set on the memory registration
• SMB Client asks the SMB Server to do a RDMA I/O on this memory registration
Memory registration in SMB2 READ

- Specifying channel
  - SMB2_CHANNEL_NONE
  - SMB2_CHANNEL_RDMA_V1
  - SMB2_CHANNEL_RDMA_V1_INVALIDATE

- SMB_DIRECT_BUFFER_DESCRIPTOR_1
  - Offset 8 bytes
  - Token 4 bytes
  - Length 4 bytes
Memory registration in Linux

Each memory registration is represented by `ib_mr`

```c
struct ib_mr {
    struct ib_device  *device;
    struct ib_pd      *pd;
    u32                lkey;
    u32                rkey;
    u64                iova;
    u32                length;
    unsigned int       page_size;
    bool               need_inval;
    union {
        struct ib_uobject *uobject;       /* user */
        struct list_head    qp_entry;       /* FR */
    }
};
```
Memory registration in Linux

Translate Linux memory registration to SMB format

struct ib_mr {
    struct ib_device *device;
    struct ib_pd *pd;
    u32 lkey;
    u32 rkey;
    u64 iova;
    u32 length;
    unsigned int page_size;
    bool need_inval;
    union {
        struct ib_uobject *uobject; /* user */
        struct list_head qp_entry; /* FR */
    };
};

SMB_DIRECT_BUFFER_DESCRIPTOR_1
- Offset 8 bytes
- Token 4 bytes
- Length 4 bytes
Memory registration in Linux

• Need to make sure memory is registered before posting the request for SMB server to initiate RDMA I/O

• FRWR (Fast Registration Work Requests)
  • Send IB_WR_REG_MR through ib_post_send
  • No need to wait for completion
  • Acts like a barrier in QP, guarantees it finishes before the following WR
  • Supported by almost all the modern RDMA hardware
Memory registration invalidation

• What to do when I/O is finished
  • Make sure SMB server no long has access to the memory region
  • Otherwise it can be messy since this is a hardware address and can be potentially changed by the server without client knowing it

• Client invalidates memory registration after I/O is done
  • IB_WR_LOCAL_INV
  • After it completes, server no longer has access to this memory
  • Client has to wait for completion before buffer is consumed by upper layer

• Starting with SMB 3.02, SMB server supports remote invalidation
  • SMB2_CHANNEL_RDMA_V1_INVALIDATE
Memory registration

SMB Client | SMB Server
---|---

Transfer I/O via Server initiated RDMA read

SMB2

Data

Wait for credits

Data

SMB Direct

Data

SMB Direct packet describing the memory location in SMB Client

Limited number of memory registration pending I/O available per QP – determined by responder resources in CM
Memory deregistration
• Need to deregister memory registration after it’s used
  • It is a slow process
  • Can slow down I/O significantly if doing it synchronously
• Maintain a list of pre-allocated memory registration slots
• Defer to background process to recover MR while other I/Os are in progress
  • Return I/O as soon as the MR is invalidated
  • How about recovery process being blocked?
  • No lock needed since there is one only recovery process modifying the list
RDMA read/write

• Overlap local CPU and communication
  • Client can proceed with other activities after memory is registered and SMB requests sent to server
  • The actually data transfer is done by RDMA hardware without CPU intervention
• Hardware figures out the best way to transfer data and handle all the I/O details. e.g. segmentation and reassembly if needed
• Reduce CPU overhead on send sider
• There is a cost for doing memory registration and invalidation
  • Suitable for larger packet
  • The default threshold is 4k bytes
    • Packets < 4k  Send/Recv
    • Packets >=4k Read/Write
Profiling

• Need to figure out the where is slow in the I/O path
• Light-weight profiling
• TSC (Time Stamp Counter)
  • Common in X86
  • BIOS for C-states
  • rdtsc()
• Store cycles in a histogram
  • Number of leading zeros in TSC cycles
  • __builtin_clzll()
RDMA Read/Write

Transfer I/O via Server initiated RDMA read

SMB Direct packet describing the memory location in SMB Client
RDMA Read/Write

SMB Client

SMB Server

SMB2

CIFS I/O thread

Process context

SMB Direct

Data

Interrupt context

Server response

Wait for credits

SMB2

Data

SMB Direct
- RDMA is completed from SOFTIRQ
- Completion Queue is polled one by one on the same CPU
- Upper layer reads I/O in a kernel thread on another CPU
- Reassembly queue is used to pass data from CPU1 to CPU2
Reduce lock contention in receive queue

- Those two CPUs don’t modify the same data
- Memory barrier is used to reduce lock contention
- Ring buffer?
RDMA send path

- Send path is about 6 - 10 times slower than receive path

CIFS code to send data:

1. allocate buffer
2. call transport to send buffer
3. release buffer

- RDMA doesn’t do buffers, so step 2 has to wait until I/O is finished
  - wait_for_completion involves calling schedule

- Solution: move CIFS buffer allocation code to SMB Direct layer
  - Estimated improvement: 5 – 20% on high queue depth
Use SMB Direct with Linux SMB Client

• mount -t -o rdma,vers=3.02

• Problems?
  • Kernel messages
  • /proc/fs/cifs/DebugData
  • /sys/module/cifs/parameters/smbd_logging_class
Test setup

- **Linux SMB Client**
  - 2 x Intel E5-2650 v3 @ 2.30GHz
  - 128 GB RAM
  - Mellanox ConnectX-3 Pro 40G Infiniband

- **Windows 2106 SMB Server**
  - 2 x Intel E5-2695 v2 @ 2.40GHz
  - 128 GB RAM
  - SMB share on RAM disk
  - Mellanox ConnectX-3 Pro 40G Infiniband

- **Switch**
  - Mellanox SX6036 in Infiniband mode

- **SMB dialect 3.02**
I/O Throughput Write

- MB/s vs. I/O size for different I/O sizes (4k, 16k, 64k, 256k, 1m, 4m) and different commands:
  - Lines: 1, 4, 16, 64, 256

Chart showing the throughput in MB/s for various I/O sizes and commands.
If we have multiple connections
2 QPs vs 1 QP

I/O size: 1m
queue depth: 16

30% increase for WRITE
Unchanged for READ
Future work

- Upper layer uses SMB Direct layer to allocate buffers
- Support multiple channel
  - Multiple QPs
Questions
Thank you