Python RDMA

Management, Diagnostics and Testing

Presented By: Jason Gunthorpe - CTO Obsidian Research

Date: OFA Monterey 2011-04-05

What is in it?

- RDMA device discovery:

```
for I in rdma.get_devices(): print I.name;
```

- RDMA Verbs:

```
with rdma.get_verbs(path.end_port) as ctx:
    print ctx.query_device();
```

- IB Management:

```
cpi = umad.SubnAdmGet(IBA.MADClassPortInfo);
```

Plus more!

What is Python?

Python is a modern, very high level, multi-paradigm programming language:

- Emphasis on readability and clarity
- Modern high level features: exceptions, garbage collection, dynamic 'duck' typing, closures
- Very popular for web development, finance, and for system administration.
- Included by default and used in major Linux distributions for many years

Cognitive Dissonance

Python is slow!

RDMA is for high performance!

Why!!?!?

Sometimes correct and simple is more important than fast..

... and good algorithms can help.

Package Contents

- RDMA Device Discovery
- Definitions from the IBA
- IB MAD RPC handling and parallelism
- IB subnet topology database
- *libibverbs* interface (*Pyrex*)
- ibtool command line program
- Codegen'd and hand written documentation
- Test suite

Pure Python except for *rdma.ibverbs*!

GPL licensed

Package Contents (2)

OFA Module	Python-rdma
libibmad	Near 100% coverage via *rdma.madtransactor* and *rdma.IBA*
libibumad	100% coverage via *rdma.umad*
libibverbs	100% coverage via *rdma.ibverbs* (through Pyrex)
libibnetdisc	~80% coverage. No support for switch chassis grouping.
librdmacm	Not covered
libibcm	Not covered
infiniband-diags	45 commands re-implemented, 2 un-implemented. Review *ibtool*
ibutils	Good coverage of the internal APIs but no coverage for the user tools.
perftest	*rdma_bw* is implemented as an example.

It works!

```
$ ibtool rdma_bw 127.0.0.1
path to peer IBPath(end_port='mlx4_0/1',
    DGID=GID('fe80::2:c903:9:1edd'),
    DLID=1, MTU=4, packet_life_time=0,
    SGID=GID('fe80::2:c903:9:1edd'),
    SLID=1, dack_resp_time=15L, dqpn=524361L,
    dqpsn=6645404, drdatomic=0,
    rate=3, sack_resp_time=15L, sqpn=524360L,
    sqpsn=1754047, srdatomic=0)
MR peer raddr=7fd268a9c000 peer rkey=8002200
1000 iterations of 1048576 is 1048576000 bytes
3065.7 MB/sec
```

MT26428 using internal loopback, 2.8GHz i5-2300

ibtool

Re-implementation of *infiniband-diags* using Python as the implementation language:

- One language
- Greater consistency
- Higher performance

Also:

- Test the Python RDMA core library
- Access the unique features of the Python RDMA via the command line
- Serve as programming examples

45 commands are implemented, > 90% complete

ibtool (2)

Mostly looks the same:

```
$ ibtool ibaddr 7
GID fe80::17:77ff:feb6:2ca4 LID start 7 end 7
$ ibtool ibswitches
Switch  : 0017:77ff:feb6:2ca4 ports 2 "Obsidian
Switch  : 0017:77ff:fef9:6e79 ports 2 "Obsidian
$ ibtool smpquery -P 2 NI -D 0,2
# Node info: DR Path (0, 2)
BaseVers:.........................1
ClassVers:........................1
NodeType:.........................2
NumPorts:.........................2
SystemGuid:.......................0017:77ff:fef9:
Guid:.............................0017:77ff:fef9:
```

ibtool (3)

Some are new:

```
$ ibtool perfquery --vl-xmit-wait  9
# Port counters: Lid 9 (fe80::17:77ff:fef9:6e79)
PortSelect:......................1
CounterSelect:..................0x0000
PortVLXmitWait[0]:..............606

$ ibtool subnet_diff ref
Current subnet has 4 end ports, reference subnet
 All end ports in the current subnet are in the
 All end ports in the reference subnet are in th
Current subnet has 3 nodes, reference subnet has
 All nodes in the current subnet are in the refe
 All nodes in the reference subnet are in the cu
```

ibtool (4)

Section 8 of the Python RDMA manual details the various differences between *ibtool* and *infiniband-diags*:

- Greater alignment with the IBA, PR usage, timeout computations, support for routed GIDs, etc
- Everything supports GID/GUID/LID/DR path as a TARGET
- Better diagnostics and debug output, including packet decodes
- --*sa* and support for GMP over verbs lets *ibtool* return info without access to /dev/umad
- LID and SA based subnet discovery options
- Consistent support for a discovery caching file

Library Tour - Device Discovery

- *rdma.devices* module - trundles through sysfs and gets devices, end ports.
- Common basis for all other modules - *umad* and *ibverbs* are all opened based on these objects.
- Find devices by string:

Format	Example
device	mlx4_0
Node GUID	0002:c903:0000:1491

- Find ports by string:

Format	Example
device	mlx4_0 (defaults to the first port)
device/port	mlx4_0/1
Port GID	fe80::2:c903:0:1491
Port GUID	0002:c903:0000:1491

Library Tour - Device Discovery (2)

Library features flow into *ibtool*:

```
$ ibtool ibaddr -P fe80::2:c903:0:14a6 9 -d
D: Using end port mlx4_0/2 fe80::2:c903:0:14a6
D: SMP Path 10 -> 9 SL=0 PKey=0xffff DQPN=0
      IBPath(end_port='mlx4_0/2', DLID=10,
             SLID=10, dqpn=0, qkey=0x0,
             sqpn=0)
D: RPC MAD_METHOD_GET(1) SMPFormat(1.1)
   SMPNodeInfo(17) completed to
   'Path 10 -> 9 SL=0 PKey=0xffff DQPN=0'
   len 256.
D: RPC MAD_METHOD_GET(1) SMPFormat(1.1)
   SMPPortInfo(21) completed to
   'Path 10 -> 9 SL=0 PKey=0xffff DQPN=0'
```

Library Tour - IBA

Structures and constants from the IBA:

- Starts out as XML describing the precise on-the-wire structure layout
- Processed via script into Python classes with *pack*, *unpack* and *printer* functions
- 106 structures from IBA
- Useful constants, value to string and string to value are hand written
- Auto generate tricky things like *SAFormat.componentMask*

Library Tour - IBA (2)

Everything can be decoded and dumped:

```
$ ibtool ibaddr 9 -dd
D: Reply MAD_METHOD_GET_RESP(129) SMPFormat(1.1)
  0 01010181 baseVersion=1,mgmtClass=1,classVers
  4 00000000 status=0,classSpecific=0
  8 000079FF transactionID=134139628569652
 12 D0E94434
   + data SMPNodeInfo
 64 01010202 baseVersion=1,classVersion=1,nodeTy
 68 001777FF systemImageGUID=GUID('0017:77ff:fef
 72 FEF96E79
 76 001777FF nodeGUID=GUID('0017:77ff:fef9:6e79'
 80 FEF96E79
 84 001777FF portGUID=GUID('0017:77ff:fef9:6e79'
```

Library Tour - IBA (3)

Dynamic language with introspection makes this dead easy:

```
$ ibtool query SubnAdmGetTable SANodeRecord \
  -f nodeInfo.systemImageGUID=0017:77ff:fef9:6e7
Reply structure #0
  LID.............................9
  nodeInfo.NumPorts...............2
  nodeInfo.SystemImageGUID........0017:77ff:fef
  nodeInfo.PortGUID...............0017:77ff:fef
  nodeInfo.VendorID...............0x001777
  nodeDescription.NodeString......'Obsidian Lon
```

45 LOC! - perform any RPC, with any arguments and pretty print the result. Widely used in implementing *ibtool*.

Library Tour - MAD Handling

- Two MAD QP interfaces - *rdma.umad* (SMP and GMP) and *rdma.vmad* (only GMP)
- Simplified programming model for issuing RPC MADs, RPC errors are converted into exceptions. checks, parsing and RMPP are centralized.
- *rdma.SATransactor* transparently converts SMP RPCs into SA RPCs - enables all tools to use *VMAD* and return data from the SA.
- *rdma.sched* parallelizes MAD RPCs - extremely easy to use, major performance win. Used extensively in *ibtool*

Library Tour - MAD Handling (2)

```
$ ibtool ibaddr 10 --sa -d
D: RPC MAD_METHOD_GET(1) SAFormat(3.2)
     SANodeRecord(17) completed to 'Path 8 -> 8
D: RPC MAD_METHOD_GET(1) SAFormat(3.2)
     SAPortInfoRecord(18) completed to 'Path 8 -
GID fe80::2:c903:0:14a6 LID start 10 end 10

$ ibtool ibnetdiscover --sa -d
D: Performing discovery using mode 'SA'
D: RPC MAD_METHOD_GET_TABLE(18) SAFormat(3.2)
    SANodeRecord(17) completed to 'Path 8 -> 8 S
D: RPC MAD_METHOD_GET_TABLE(18) SAFormat(3.2)
    SAPortInfoRecord(18) completed to 'Path 8 ->
D: RPC MAD_METHOD_GET_TABLE(18) SAFormat(3.2)
    SALinkRecord(32) completed to 'Path 8 -> 8 S
```

Library Tour - MAD Parallelism

Python Co-Routines - one thread, multiple execution contexts:

```
def get_pinf(sched,path,idx):
  pinf = yield sched.SubnGet(IBA.SMPPortInfo,
                             path,idx);
sched.mqueue(get_pinf(sched,path,idx)
        for I in range(1,ninf.numPorts+1));
```

Run *numPorts* copies of *get_pinf* in parallel. Automatically limits outstanding RPCs, tracks completion, manages timeouts, etc.

Library Tour - IB Subnet

Fetch, store and manipulate an IB subnet:

- Discovery via DR SMP, LID SMP or SA *SubnAdmGetTable*
- Incremental out of order loading
- Save/Load to a Python pickle
- Iterate, BFS iterate, lookup by GUID, etc.

Library Tour - IB Subnet (2)

All *ibtool* discovery using functions support common options and caching:

```
$ ibtool ibnetdiscover --cache disc \
         --refresh-cache
$ ibtool ibcheckerrors --cache disc
## Summary: 4 nodes checked, 0 bad nodes found
##          8 ports checked, 0 ports with bad st
##          4 ports checked, 0 ports have errors
```

No MADs will be issued by *ibcheckerrors*

Library Tour - Verbs

Easy to use wrappers around verbs:

```
with get_verbs(path.end_port) as ctx:
    cq = ctx.cq(100,ctx.comp_channel());
    pd = ctx.pd();
    qp = pd.qp(ibv.IBV_QPT_UD,100,100,cq);
```

- Errors are raised as exceptions
- *libibverbs* functions cast into objects
- Reference counting and Python context managers ensure correct resource cleanup

Library Tour - Verbs (2)

Simplifications for WC processing:

```
poller = CQPoller(cq);
for wc in poller.iterwc(timeout=1):
 if wc.status != ibv.IBV_WC_SUCCESS:
   raise ibv.WCError(wc,cq,obj=qp);
```

- Iterate over WC's, block with *poll*
- Transparently handle async events
- Place a timeout around the entire for loop
- Messy details to prevent races are hidden
- WC errors raise as exceptions and pretty print

Library Tour - Verbs (3)

Tight integration with *IBPath* concept:

```
path = get_mad_path(umad,"10");
qp.establish(path);
qp.post_send(ibv.send_wr(
    opcode=ibv.IBV_WR_SEND,ah=pd.ah(path),
    remote_qpn=path.dqpn,remote_qkey=path.qkey));
```

- Caches AH construction
- Verbs modify_qp draws information from the path (eg pkey, qkey, psn, etc)
- Works for UD, UC and RC,
- Can also get a path from a WC

Summary

- Great for writing management tools
- Very time efficient for test development, training and prototyping
- *ibtool* is an improved, simpler and more maintainable version of the diags programs

Thanks!

Get it at GitHub

http://github.com/jgunthorpe/python-rdma

Read the manual!

Try it out!