PASTE: A Network Programming Interface for Non-Volatile Main Memory

Michio Honda (NEC Laboratories Europe)
Giuseppe Lettieri (Università di Pisa)
Lars Eggert and Douglas Santry (NetApp)

IIJ lab Seminar, May 2nd 2018, Tokyo, Japan
Summary

● PASTE is a network programming interface that:
  ○ Enables zero copy to NVMM
  ○ Helps apps organize persistent data structures on NVMM
  ○ Lets apps use modern TCP/IP and be protected
  ○ Offers high-performance network stack even w/o NVMM

https://github.com/luigirizzo/netmap/tree/paste
Review: Memory Hierarchy

Slow, block-oriented persistence

- **CPU Caches**: 5-50 ns
- **Main Memory**: 70 ns
- **HDD / SSD**: 100-1000s us

Access:
- **Byte access with load/store**: 100-1000s us
- **Block access with system calls**: 70 ns
Review: Memory Hierarchy

Fast, byte-addressable persistence

- **CPU Caches**: 5-50 ns
- **Main Memory**: 70 ns - 1000s ns
- **HDD / SSD**: 100-1000s us

- **Byte access w/ load/store**
- **Block access w/ system calls**
Networking is faster than disks/SSDs

1.2KB durable write over TCP/HTTP

Client → Server

Cables, NICs, TCP/IP, socket API

23us

Server → SSD

Syscall, PCIe bus, physical media

1300us
Networking is slower than NVMM

1.2KB durable write over TCP/HTTP
Networking is slower than NVMM

1.2KB durable write over TCP/HTTP

```
nevts = epoll_wait(fds)
for (i =0; i < nevts; i++) {
    read(fds[i], buf);
    ...  
    memcpy(nvmm, buf);
    ...
    write(fds[i], reply)
}
```
Innovations at both stacks

Network stack
- MegaPipe [OSDI'12]
- Seastar
- mTCP [NSDI'14]
- IX [OSDI'14]
- Stackmap [ATC’16]

Storage stack
- NVTree [FAST’15]
- NVWal [ASPLOS’16]
- NOVA [FAST’16]
- Decibel [NSDI’17]
- LSNVMM [ATC’17]
Stacks are isolated

**Network stack**
- MegaPipe [OSDI’12]
- Seastar
- mTCP [NSDI’14]
- IX [OSDI’14]
- Stackmap [ATC’16]

**Costs of moving data**

**Storage stack**
- NVTree [FAST’15]
- NVWal [ASPLOS’16]
- NOVA [FAST’16]
- Decibel [NSDI’17]
- LSNVMM [ATC’17]
Bridging the gap

Network stack
- MegaPipe [OSDI’12]
- Seastar
- mTCP [NSDI’14]
- IX [OSDI’14]
- Stackmap [ATC’16]

PASTE

Storage stack
- NVTree [FAST’15]
- NVWal [ASPLOS’16]
- NOVA [FAST’16]
- Decibel [NSDI’17]
- LSNVMM [ATC’17]
PASTE Design Goals

- Durable zero copy
  - DMA to NVMM

- Selective persistence
  - Exploit modern NIC’s DMA to L3 cache

- Persistent data structures
  - Indexed, named packet buffers backed by a file

- Generality and safety
  - TCP/IP in the kernel and netmap API

- Best practices from modern network stacks
  - Run-to-completion, blocking, busy-polling, batching etc
PASTE in Action
PASTE in Action

App thread

Zero copy

Pring

Plog

/mnt/pm/plog

User

Kernel

NIC

TCP/IP

File system

/mnt/pm

Ppool (shared memory)

/mnt/pm/pp

Pbufs

Slot [0] [7]

Cur [0] [4] [8]

[21 22 23 24 25 26 27]
PASTE in Action

- poll() system call

1. Run NIC I/O and TCP/IP

- App thread
- Zero copy

- Pring
- Ppool (shared memory)

- File system

- NIC
- TCP/IP
PASTE in Action

1. Run NIC I/O and TCP/IP

- poll() system call
  - Got 7 in-order TCP segments

Zero copy

App thread

File system

/misc/pm

Zero copy

User

Kernel

Ppool (shared memory)

/misc/pm/whatever

Pbufs

TCP/IP

NIC
PASTE in Action

- poll() system call
  - They are set to Pring slots
PASTE in Action

1. Run NIC I/O and TCP/IP

- Return from poll()
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on Pring

App thread

Zero copy

Pring

Plog
/mnt/pm/plog

<table>
<thead>
<tr>
<th>pbuf</th>
<th>off</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ppool (shared memory)
/mnt/pm/pp

File system
/mnt/pm

TCP/IP

NIC

slot [0]
cur

[7]
tail

Pbufs

[0]

[4]
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on Pring
3. Flush Pbuf(s)

- flush Pbuf data from CPU cache to DIMM
  - clflush(opt) instruction

[Diagram showing app thread, Pring, Ppool, Plog, and File system]
PASTE in Action

- Pbuf is persistent data representation
  - Base address is static i.e., file (/mnt/pm/pp)
  - Buffers can be recovered after reboot

1. Run NIC I/O and TCP/IP
2. Read data on Pring
3. Flush Pbuf(s)
4. Flush Plog entry(ies)

Zero copy

Pbufs

 NIC
 TCP/IP
/nt/pm/pp
Ppool (shared memory)
/mnt/pm/pp
Pring
/mnt/pm/plog

File system
/mnt/pm

App thread

User

Kernel

cur

0 1 2 3 4 5 6 7 tail

Slot

Plog

<table>
<thead>
<tr>
<th>pbuf</th>
<th>off</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
<td>120</td>
</tr>
</tbody>
</table>

len
off
pbuf

Zero copy

TCP/IP
NIC

27
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on Pring
3. Flush Pbuf(s)
4. Flush Plog entry(ies)
5. Swap out Pbuf(s)

- Prevent the kernel from recycling the buffer

 zero copy

App thread

NIC
TCP/IP

File system
/mnt/pm/pp

Pring

Ppool (shared memory)

Plog

/mnt/pm/plog

user

kernel

Slot [0] [7] cur

[0] [4] [8] tail Pbufs

Pbuf off len

1 96 120

File system
/mnt/pm

TCP/IP

NIC
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on Pring
3. Flush Pbuf(s)
4. Flush Plog entry(ies)
5. Swap out Pbuf(s)

- Same for Pbuf 2 and 6
PASTE in Action

- Advance cur
  - Return buffers in slot 0-6 to the kernel at next poll()
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on Pring
3. Flush Pbuf(s)
4. Flush Plog entry(ies)
5. Swap out Pbuf(s)
6. Update Pring

Write-Ahead Logs

user

kernel

Zero copy

Ppool (shared memory)
/mnt/pm/pp

File system
/mnt/pm

TCP/IP

NIC

/mnt/pm/plog

<table>
<thead>
<tr>
<th>pbuf</th>
<th>off</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>768</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>987</td>
</tr>
</tbody>
</table>

Pring

[0] 0 27 10
tail
cur

[7] 8 9 3 4

Pbufs

[0] [4] [8]

Tail

cur

Slot [0]
We can organize various data structures in **Plog**
Evaluation

1. How does PASTE outperform existing systems?
2. Is PASTE applicable to existing applications?
3. Is PASTE useful for systems other than file/DB storage?
How does PASTE outperform existing systems?

What if we use more complex data structures?
How does PASTE outperform existing systems?

**WAL**

- **64B**
  - Throughput [K trans/s]
  - Latency [µs]

- **1280B**
  - Throughput [K trans/s]
  - Latency [µs]

**B+tree (all writes)**

- **64B**
  - Throughput [K trans/s]
  - Latency [µs]

- **1280B**
  - Throughput [K trans/s]
  - Latency [µs]
Is PASTE applicable to existing applications?

- Redis
Is PASTE useful for systems other than DB/file storage?

- Packet logging *prior to forwarding*
  - Fault-tolerant middlebox [Sigcomm’15]
  - Traffic recording
- Extend mSwitch [SOSR’15]
  - Scalable NFV backend switch
Multicore Scalability

- WAL throughput
Conclusion

- PASTE is a network programming interface that:
  - Enables durable zero copy to NVMM
  - Helps apps organize persistent data structures on NVMM
  - Lets apps use modern TCP/IP and be protected
  - Offers high-performance network stack even w/o NVMM

https://github.com/luigirizzo/netmap/tree/paste
micchie@sfc.wide.ad.jp or @michioh
Further Opportunity with Co-designed Stacks

- What if we use higher access latency NVMM?
  - e.g., 3D-Xpoint
- Overlap flushes and processing with clflushopt and mfence before system call (triggers packet I/O)
  - See the paper for results
Experiment Setup

- Intel Xeon E5-2640v4 (2.4 Ghz)
- HPE 8GB NVDIMM (NVDIMM-N)
- Intel X540 10 GbE NIC

Comparison
  - Linux and Stackmap [ATC’15] (current state-of-the-art)
  - Fair to use the same kernel TCP/IP implementation
PASTE Concept

- **DMA to NVMM, just flush if necessary**
Discussion and Future Work

- Encrypted data
  - Modern/Smart NICs
- Impact on NVMM wear
- In the paper
  - How to cope with higher NVMM access latency
    - e.g., Intel 3D-Xpoint
PASTE Concept

- **DMA to NVMM, just flush if necessary**
PASTE Concept

- **DMA to NVMM, just flush if necessary**

Write-ahead logging performance for 1.2 KB data
Is PASTE useful to systems other than DB/file storage?

- Packet logging *prior to forwarding*
  - Fault-tolerant middlebox [Sigcomm’15]
  - Traffic recording
- Extend mSwitch [SOSR’15]