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)

USENIX NSDI 2018
Review: Memory Hierarchy

Slow, block-oriented persistence
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**: 70 ns - 1000s ns
- **Block access w/ system calls**: 100-1000s us
Networking is faster than disks/SSDs

1.2KB durable write over TCP/HTTP

Client

Cables, NICs, TCP/IP, socket API

Server

Syscall, PCIe bus, physical media

23us

SSD

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]

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

PASTE
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

- **App thread**
  - **Zero copy**

- **Pring**
  - **Ppool** (shared memory)
    - **Pbufs**

- **Plog**
  - `/mnt/pm/plog`
    - `pbuf` | `off` | `len`

- **File system**
  - `/mnt/pm`
PASTE in Action

- poll() system call

1. Run NIC I/O and TCP/IP

App thread

Zero copy

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>

user

Ppool (shared memory)
/mnt/pm/pp

slot [0] Pring [7]

/ m n t / p m / p p

File system
/mnt/pm

TCP/IP

NIC

File system
/mnt/pm

TCP/IP

NIC

Pbufs
PASTE in Action

- poll() system call
  - Got 6 in-order TCP segments
PASTE in Action

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

- Return from poll()

1. Run NIC I/O and TCP/IP

- Zero copy

- App thread

- Plog
  - /mnt/pm/plog
  - pbuf
  - off
  - len

- Pring
  - /mnt/pm/pp
  - slot [0]
  - cur [7]
  - tail

- Ppool (shared memory)
  - /mnt/pm/pp
  - [0]
  - [4]
  - [8]
  - Pbufs
  - **...

- File system
  - /mnt/pm
  - TCP/IP
  - NIC

- Zero copy

- User

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

Zero copy Pbufs
PASTE in Action

- Pbuf is a 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)
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

App thread

NIC

TCP/IP

File system

Zero copy

Pbuf

Plog

/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>
</tbody>
</table>

Ppool (shared memory)

/mnt/pm/pp

[0] Pbufs

Pring

/mnt/pm/pring

[0] 27 6
[2] 27 6
[3] 27 6
[4] 27 6
[5] 27 6
[6] 27 6
[7] 27 6
[8] 27 6

user

kernel

8 27 6 5 4 3 2 1

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

File system
/mnt/pm
TCP/IP
NIC
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?

- **64B**
  - WAL
  - B+tree (all writes)

- **1280B**
  - WAL
  - B+tree (all writes)
Is PASTE applicable to existing applications?

- Redis

![Graph showing throughput for different write ratios and workloads with PASTE and YCSB compared to Linux and Linux (no persist).]
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
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 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
Multicore Scalability

- WAL throughput
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