NeVer Mind networking: Using shared non-volatile memory in scale-out software

Stanko Novakovic*, Paolo Faraboschi, Kimberly Keeton, Rob Schreiber, Edouard Bugnion*, Babak Falsafi*

EPFL, Switzerland* HP Labs, USA
Multiple non-volatile memory technologies

**Magnetic disk**
- Block-based storage, access via programmed IO or DMA (e.g. ATA)

**Flash (PCIe-attached – via NVMe)**
- Block-based storage, access via queue-pairs (i.e. SQ, CQ) and DMA

**Non-volatile Random Access Memory (NVRAM)**
- Persistent with similar performance characteristics to DRAM
- Byte-addressable, direct access via LD/ST
- Examples: Resistive RAM (RRAM), Phase-change memory (PCM)
Rack-scale systems with shared non-volatile memory

<table>
<thead>
<tr>
<th></th>
<th>Shared disk architecture</th>
<th>Shared NVMe namespaces</th>
<th>TheMachine/Firebox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache coher.</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Interface</td>
<td>PIO, DMA</td>
<td>QP-based</td>
<td>LD/ST</td>
</tr>
<tr>
<td>Technology</td>
<td>Magnetic disk</td>
<td>Flash</td>
<td>NVRAM</td>
</tr>
</tbody>
</table>

1. Direct shared access  
2. Latency: small factor of DRAM  
3. Bandwidth: DDR bandwidth
The Machine from HP Labs

Petabytes of byte-addressable NVRAM based on memristor technology
  - Shared across thousands of cores via photonic interconnect

High-performance, high-capacity rack-scale computing
Outline

Introduction to shared NVRAM

→ Shared NVRAM model

Hash Table

Graph Processing System

Conclusion
Shared NVRAM model

Servers
- Do not share DRAM
- Share NVRAM
  * but not CC *

Shared NVRAM
- Used as a heap
- Latency: 4-5x of DRAM
- Bandwidth: DDR rate

Can datacenter software benefit from such architecture?
Experimental shared NVRAM platform

Kernel Virtual Machines (KVM)

Fraction of each guest’s address space is shared

User code instrumented to delay NVRAM accesses

\[ T_{\text{trans}} = \text{lat} + \frac{\text{data size}}{\text{bandwidth}} \]

\[ \approx 0.5\mu s \]

DDR3/QPI (unchanged)
Using shared NVRAM in datacenter software systems

Data serving
- Distributed hash tables, graph stores  
  ➔ Share data items via NVRAM

Data processing (a.k.a. analytics)
- MapReduce, distributed graph processing, etc.  
  ➔ Use shared NVRAM as communication medium
Today: Hash Table w/ client-side hashing

client

shardID ← h(key)…
This work: CREW Hash Table for shared NVRAM

Each server owns write permission to one block and read permission to whole NVRAM

Permissions:
Read-Write = \{1\}
Read-Only = \{1,2,..N\}

Data stored in NVRAM

CREW – Concurrent Read Exclusive Write
CREW Hash Table for shared NVRAM (reads)

client

serverID = uniform_rand()
CREW Hash Table for **shared NVRAM (writes)**

```
client

serverID = uniform_rand()
```

Diagram showing a network connection between a client and multiple CPU/DRAM nodes, each connected through a Memory Interconnect.
Hash table for shared NVRAM prototype

Based on Redis KVS
- Each server runs separate instance of Redis
- Shared read-only access to data items

YCSB clients run on separate servers
- Configured to compute hash or pick server in round-robin fashion
Per-server network utilization (0.99 Zipf workload)

Consistent hashing on client side can cause load imbalance
What does uniform utilization buy us?

99th-percentile latency

Higher throughput due to shared access w/o violating SLA
Today: Hash Table w/ **proxying**

client

serverID = uniform_rand()
HT w/ proxying vs. HT w/ shared access

- ~2x improvement in performance for read-only workload
Data serving recap

Looked at two hash table (KVS) designs
• with client-side hashing and proxying

KVS that uses shared NVRAM allows for shared read-only access
• Better load balancing over hashing
• Lower end-to-end latency over proxying
Outline

Introduction to shared NVRAM

Shared NVRAM model

Hash Table

→ Graph Processing System

Conclusion
Bulk Synchronous Parallel (BSP) on scale-out

Buffered vertex updates
BSP graph processing on **scale-out**

1. CPU
2. CPU
\[ \text{network} \]

I/O

N
CPU

Compute
Communicate
BSP graph processing on shared NVRAM
BSP graph processing on **shared NVRAM**

![Diagram of BSP graph processing on shared NVRAM](image)

- **Compute**
- **Communicate**
- **DRAM → NVRAM**

**Memory Interconnect**

**FLUSH**
BSP graph processing on shared NVRAM

Compute
Communicate
DRAM \rightarrow NVRAM
NOTIFY

Memory Interconnect
BSP graph processing on **shared NVRAM**

1. **CPU**
2. **CPU**
   - **PULL**
   - **Communicate**
   - **DRAM → NVRAM**
   - **NOTIFY**
   - **NVRAM → DRAM**

**Compute**

**Memory Interconnect**
BSP graph processing for **shared NVRAM** prototype

- Shared NVRAM BSP framework implemented in C++ from scratch
- Algorithms implemented as compute kernels
- Compare BSP over TCP/IP and BSP over NVRAM
PageRank on Twitter graph

Accelerating shuffle phase helps improve overall execution time
Related work

Concurrent Read Exclusive Write (FARM NSDI’14, MICA NSDI’14)

Graph update aggregation (Pregel SIGMOD’10)

Shared disks and shared NVMe namespaces
Conclusion

Rack-scale systems w/ shared NVRAM
• Direct access, byte-addressable

This talk: how to make use of this arch.
• In common DC apps
• Studied KVS and graph processing
Thank you