The Future of Memory

SECDL Retreat 05/28/15

Prof. William Dally
NVIDIA/Stanford University

Prof. Christos Kozyrakis
Stanford University
Physical Structure of Memory

Node

Chip

Core

P

L1

SP

1cy 1pJ

3cy 10pJ

10cy 30pJ

30cy 200pJ

L2

SP2

LLC

LSP

150cy 1nJ

1E7cy 5mJ

50Kcy 10uJ

Node

1000cy 5nJ

Node

Disk

DRAM

NV

DRAM

NV

DRAM

NV
Logical View of Memory

<table>
<thead>
<tr>
<th>Table</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>3</td>
</tr>
<tr>
<td>Friends</td>
<td>8</td>
</tr>
<tr>
<td>Loc</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>
Mismatch Between

- Physical structure of memory
- Logical view of memory
- Architecture and mechanisms that expose memory
Obsolete Assumptions

• NV storage is a very slow block device
• Apps exhibit high temporal locality
• Move data to the compute

• No longer the case
  – Apps with massive, in-memory datasets
    • KV stores, graph systems, No-SQL databases, search …
    • Limited locality as they serve requests from millions of users
  – Storage class memories (SCM): fast & byte-level
    • PCM, resistive RAM
  – Moving computation to the data
    • For energy efficiency
The Requirements

- Need to rethink HW & SW for memory systems

- Features needed
  - Very high capacity memory systems
    - All data in some kind of fast memory
  - Support for heterogeneity: DRAM + SCM
    - And a deep hierarchy
  - Support for aggressive sharing
    - Between applications on a single node
    - Between nodes in a rack (disaggregation)
  - Near data processing
  - Security and privacy
Files & Pages Will not Cut It

- **Pages**
  - Wrong granularity
    - Too coarse for data movement
    - Too fine for protection and translation
  - Inflexible
  - Difficult to share (pointers in different address spaces)

- **Files**
  - Very heavyweight abstraction
  - Overheads of serialization/deserialization
Towards a One-Level Store

- Single address space spanning
  - Levels of storage (cache to NV)
  - Nodes

- No need to translate pointer structures
  - From file to DRAM
  - From node to node
Segments with Attributes

• A segment represents a set of records
  – Table, list, etc…

• Segments have “attributes”
  – (see next slide)

• Pointers identify a record within a segment
Attributes

• Logical attributes:
  – Size
  – Persistence
  – Durability
  – Mutability
  – Sparsity
  – Managed
  – Indexed
  – Locality patterns
  – …

Users specify logical attributes

• Physical attributes:
  – Location(s)
  – Representation (encoding)
  – Distribution
  – Replication
  – Encryption
  – Compression

Mappers determine physical attributes

Hints may be provided to help the mappers
Basic Operations

- **seg = create_segment(rows, cols, persistent, sparse)**;  
  - Sparse segments require ranges of rows to be materialized
- **free_segment(seg)**;  
  - May be implied by scoping, or automatic via GC
- **rec = get_record(seg, offset)**
- **value = get_field(seg, offset, field)**  
  - Load
- **set_field(seg, offset, field, value)**  
  - Store
- **get_indexed(seg, field, value)**
Implications

• A file is just a persistent segment
  - It may be bound to a name in a directory
• A database is just a set of persistent, durable segments
• A key-value store is just a persistent, durable, indexed segment
• Segments may be shared across nodes and between processes within a node
• Segments may be mapped across nodes and over levels of storage within a node
• Parts of segments may be replicated, compressed, not there, etc…
Scenarios

• A process can create a segment larger than the DRAM on its node
  – Disaggregated memory

• Pointer structures in files can just be used
  – No need to remap between two address spaces

• Persistent segments are automatically mapped to NV memory and/or disk
  – But cached in DRAM, SRAM, or SCM when in use

• Durable segments are automatically replicated as needed
Mapping

• Mapping policy decides where portions of a segment are located and replicated.

• Access mechanism
  – Validates access
  – Locates nearest copy for reads
  – Maintains coherence and consistency for writes
  – Fast (hardware) access to near levels of storage
  – Flexible (software) access to others
Just say No to Pages

• Too fine a granularity for mapping
  – 256M 4K pages in a 1TB memory system
  – Huge page tables with redundant information
  – TLB thrashing

• Too large a granularity for communication
  – Want to send 8-64B, not 4K

• Doesn’t handle replication well
Some Research Questions

• Mapping policies and mechanisms
  – In a single node, in a rack, in a datacenter
• API to provide hints
• Division of mechanism between HW, runtime, compiler
• Pointer compression
• Compilation (JIT) to remove translation
• Resilience mechanisms
• Efficient hardware structures and mechanisms
• Capabilities
• Privacy and security (flow tracking & control)
• Integration with distributed execution
  – Load balancing, near data processing, …
• Support for dynamic types