A GENERIC DURABLE OBJECT MODEL WITH DURABLE COMPUTING

Gary Wang – Intel
Johnu George, Debo Dutta – Cisco
CURRENT BIG DATA SYSTEM STACK

Real questions?
1. How to represent object graphs in software efficiently?
2. How to map software models to hardware efficiently?
PROBLEM: OBJECT TRANSFORMATIONS -> HIGH SERDES COST
PROBLEM: LIMITED HEAP SPACE
PROBLEM: HIGH MEMORY MANAGEMENT OVERHEADS
PROBLEM: HIGH COMMUNICATION OVERHEADS
SOLUTION?: LEVERAGE SMART HARDWARE
TLDR: CURRENT BIGDATA WORKLOADS

- High JVM performance costs
  - Serialization/Deserialization
  - JNI invocation cost

- High programming complexity
  - Custom caching schemes
  - Custom memory management schemes
  - Custom transformation schemes

- Poor Performance of JVM based stacks
  - Higher memory usage
  - Frequent GC pauses
Evolving new storage technologies – NVM, 3D Xpoint
- Persistent byte addressable memory speed store!
- Relative performance of system layers changed
- Changes required in software programming models

The most popular PL among Apache projects - Java
- Approx. 60% of TLP are java based projects

Need for a Java based non-volatile programming model that provides an unified view of storage technologies – DRAM, PMEM, SSD etc.

=> Mnemonic
WHAT IS MNEMONIC?

Object Graph

Objects

Objects

Compute

CPU

GPU

TPU

Storage

Generic Durable Computing Model

Generic Durable Object Model

DRAM

PMEM

SSD
GENERIC DURABLE OBJECT MODEL

- Simple but not simplistic
- Generic
- Durable
- Auto-Reclaimable
- Lazily Restorable
- Memory Agnostic
- No SerDes
- Native Identifiable
- Optional Handler Store

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Volatile</th>
<th>Object</th>
<th>Generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NonVolatile</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
AVAILABLE MEMORY SERVICES

• NVML-VMEM
• PMALLOC
• NVML-PMEM
• SYS-VMEM
BENEFIT: IN PLACE UPDATES
BENEFIT: AUTO RECLAIM

- Unreachable Durable Object
  - If the flag of autoreclaim is true
    - Used Mem. Space
  - Manually Destroy
- Garbage Collection
BENEFIT: LAZILY RESTORABLE
BENEFIT: MEMORY AGNOSTIC
CONSISTENCY

• None
• Close
• Flush
• Single Object Persist
• Object Deep Persist
• Atomic Operation
• Transaction
DURABLE OBJECT STATES

Create → Floating
Create → Restored
Create → Invisible
Create → Dangled
Create → Not existed

Floating → set autoreclaim
Floating → unset autoreclaim
Floating → Retentive
Floating → Dangled
Floating → Invisible

Retentive → set autoreclaim
Retentive → unset autoreclaim
Retentive → collect
Retentive → manually destroy
Retentive → Dangled
Retentive → Invisible

Dangled → set autoreclaim
Dangled → unset autoreclaim
Dangled → collect
Dangled → manually destroy
Dangled → Floating
Dangled → Invisible

Invisible → restore
Invisible → Invisible
Invisible → Dangled
Invisible → Invisible

Not existed → collect
Not existed → Floating
Not existed → Retentive
Not existed → Dangled
Not existed → Invisible
Not existed → Invisible
Not existed → Invisible
Not existed → Invisible

Unreachable
DURABLE OBJECT GRAPH EXAMPLE

Traditional Object Graph View (on Heap)
Operation: Search Alice’s team member Ken

Mnemonic Non-volatile Object Graph View (on Heap)
Operation 1: Search Alice’s team member Ken

Operation 2: Update Bob’s info

Possible Storage Media:
- Off-Heap Memory
- Persistent Memory
- NVMs/SSD

Load on demand
- Reclaim once used
- Share for multiple processes
- In-place creation & update
- No need to normalize
- Hybrid Memory-like device backed
- Pluggable customized allocators as service

Process Communication
- Normalize
- Move
- Join
- Index
- Search
- Type Mapping
- SerDe

Sequential Read/Write
- Move
- Cache
- Pack & Unpack
- Load all or none
- Bursty traffic
- break encapsulation
- involve filesystem Cache
DURABLE COLLECTIONS

• Simple Memory Abstraction for Common Data Structures
  • DurableChunk
  • DurableBuffer
  • DurableList
  • DurableMap
  • DurableTree
  • DurableArray
  • DurableSet (WIP)

• Distributed Collections (WIP)
EXAMPLES

Durable Maps

DurableHashMap<String, Integer> map = DurableHashMapFactory.create(allocator);
map.put("hello", 1);
map.put("world", 2);
map.put("hello", 1);

Durable Arrays

DurableArray<String> array = DurableArrayFactory.create(allocator,100);
array.set(1, "string0");
array.get(1);
DURABLE COMPUTING MODEL

- Simple but not simplistic
- Across VM boundary
- No Marshal/Un-marshalling
- Shared Viewpoint
- Schema-less
- Enable Smart Hardware
DURABLE COMPUTING SERVICE
COLLECTED OBJECTS

Path = 2 -> 1 -> 5
MNEMONIC INTEGRATION
MNEMONIC – HADOOP

MnInputFormat → MR-like Engines → MneOutputFormat

MneMapreduceRecordReader

MneMapreduceRecordWriter

Partitions 1,2,....
MNEMONIC - SPARK

makeDurable() with transform

RDD[T1] → DurableRDD[D] → RDD[T2]

Partitions 1, 2, ....
REDUCE GC ACTIVITIES

<table>
<thead>
<tr>
<th>Exp: GC Sum of executors</th>
<th>Default</th>
<th>Mnemonic</th>
<th>Mnemonic/Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of FullGC</td>
<td>23</td>
<td>12</td>
<td>0.522</td>
</tr>
<tr>
<td>Total FullGC User (s)</td>
<td>1716.82</td>
<td>800.06</td>
<td>0.466</td>
</tr>
<tr>
<td>Total FullGC Sys (s)</td>
<td>13.67</td>
<td>6.52</td>
<td>0.477</td>
</tr>
<tr>
<td>Total FullGC Real (s)</td>
<td>129.36</td>
<td>59.52</td>
<td>0.460</td>
</tr>
<tr>
<td>Total number of YoungGC</td>
<td>1310</td>
<td>864</td>
<td>0.660</td>
</tr>
<tr>
<td>Total YoungGC User (s)</td>
<td>6799.45</td>
<td>3892.05</td>
<td>0.572</td>
</tr>
<tr>
<td>Total YoungGC Sys (s)</td>
<td>833.54</td>
<td>740.81</td>
<td>0.889</td>
</tr>
<tr>
<td>Total YoungGC Real (s)</td>
<td>225.98</td>
<td>158.79</td>
<td>0.703</td>
</tr>
<tr>
<td>Total GC pauses (s)</td>
<td>355.34</td>
<td>218.31</td>
<td>0.614</td>
</tr>
</tbody>
</table>

Lesser GC for Mnemonic applications
BUBBLE SORTING EXPERIMENTS

Sorting Benchmark

- Input-5000
  - Without Mnemonic: 0
  - With Mnemonic: 0
- Input-10000
  - Without Mnemonic: 1000
  - With Mnemonic: 1000
- Input-20000
  - Without Mnemonic: 3000
  - With Mnemonic: 3000
- Input-40000
  - Without Mnemonic: 5000
  - With Mnemonic: 5000
CONCLUSION

Without Mnemonic

- High JVM performance costs
  - SerDes
  - JNI invocation
- High programming complexity
  - Own caching schemes
  - Own memory management schemes
  - Own structure transformations
- Poor Performance of JVM based stacks
  - Higher memory usage
  - Frequent GC pauses

With Mnemonic

- Durable Object Model
  - Unified view of storage devices
  - No SerDes cost
  - Less memory usage
  - Reduced JC pauses
  - Handles memory management
- Durable Computing model
  - Bypass JNI calls
  - Better performance
Thanks!

Join Mnemonic community

device-subscribe@mnemonic.incubator.apache.org

Gang (Gary) Wang
Gang1.wang@intel.com
garyw@apache.org

Johnu George
johnugeo@cisco.com
johnu@apache.org

Debo Dutta
dedutta@cisco.com
ddutta@apache.org