Boost Spark ML Performance with Project Mnemonic
Who

Yanping Wang

- Software Performance Engineer at Intel
- Apache Mnemonic (incubating) project leader
- System performance modeling
- Java runtime and GC tuning expert
- Big Data applications’ performance analysis and tuning

Gang (Gary) Wang

- Software Development Engineer at Intel
- Apache Mnemonic (incubating) project initiator
- Designs and implements Big Data applications
- Built up an integrated real-time CDR processing and massive collaborative analytics system for telecom industry
Storage Media Evolution is Changing Future SW Programing

For decades we've had two primary types of memories in computers: DRAM and Hard Disk Drive (HDD).
- DRAM was fast and volatile and HDDs were slower, but persistent (aka nonvolatile).
- Data moves from the HDD to DRAM over a bus where it is the fed to the processor.
- The processor writes the result in DRAM and then it is stored back to disk to remain for future use.
- HDD is 100,000 times greater latency than DRAM (!)

Pictures are from “Building the data infrastructure of the future with persistent memory” discussion from Strata Hadoop World in San Jose 3/30/2016
Storage Media Evolution is Changing Future SW Programming

Mnemonic is a pioneer project to extend **in-memory** storage and computing to **in-Place** storage and computing on next generation NVM storage media. (Pictures were taken from Storage Industry Summit January, 20, 2016)
Non-Volatile Java Object Model

- Provides java programmers a mechanism to define objects and structures to be Mnemonic managed objects and structures.
- Objects that identified as @DurableEntity store their durable fields in volatile/non-volatile memory space.
- @DurableGetter and @DurableSetter are used to access those fields.
- Handlers are used to refer their connected durable object.
Traditional Java based Big Data Frameworks and Applications

0: Start with Clean Java Heap

1: Deserialize Objects from media into Memory

2: Process in Heap Memory

3: Serialize Objects to media (or discard)

4: Resulting Java Heap (in progress)

Problems
- Heap “Garbage”, Memory Pressure!
- “SerDes” costs can be significant
Apache Mnemonic: Leave data in-place

For Big Data Computation, keep massive data object and its schema on storage. No need to serialize/deserialize the objects.

1: extract the bootstrap handlers from Key-Value store. the values & references of data objects were held in place on the underlying storage (e.g. PM)

3: Data objects are loaded & visited on demand. It is not necessary to hold them on memory after use

Benefits
- Less Garbage, Less Memory Pressure!
- No “SerDes” costs
A Mnemonic Search 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

- Normalized
- Move
- Join
- Index
- Search
- Type Mapping
- Sort

- Move
- Cache
- Pack & Unpack
- Load all or none
- Bursty traffic
- break encapsulation
- Incomplete cache

Possible Storage Media
- Off-Heap Memory
- Persistent Memory
- NVMe/SSD

- Load on demand
- Reclaim once used
- Share for multiple processes
- In-place creation & update
- No need to normalize
- Hybrid Memory-like device backed
- Preglance customized allocators as services
Apache Mnemonic (Incubating)

• Apache Mnemonic is a Java based non-volatile memory library for in-place structured data processing and computing. It provides unified interfaces for data manipulation on heterogeneous block/byte-addressable devices, include DRAM, persistent memory (NVDRAM), NVMe, 3DX SSD.

• Addresses Big Data performance issues
  • High cost of Serialization/De-serialization
  • Frequent unpredicted long Garbage Collection pauses
  • Lack of caching, burst temporary object creation/destruction
  • Frequent high cost JNI call with stack marshalling
  • System paging and kernel structure updating due to massive data processing
Apache Mnemonic Infrastructure

System Technologies & Optimization (STO)

Durable Objects

Memory Serialization

Java Serializable Objects

Java Byte Array like Data

Java Unsafe

Scalable Memory Resources

Memory Buffer

Java Data Structures, e.g. Map, List, Set

Memory Chunk

Memory Cache Pool

NonVolatile Memory Cache Pool

Memory Clustering

SysMem Allocator

VolatileMem Alloc. Service

NonVolatile Mem Alloc. Service

Eviction Filter
Drop Event

Eviction Filter
Drop Event

Perf. Change Event
Allocate
Change Event

DRAM

SSD, NVMe

SSD, NVMe, NAS

Compression

Serialization

Crypto

Intel On-Chip Speedup
Mnemonic Structure

• Performance oriented architecture
• Unified platform enabling Framework
• Unique non-volatile object model
• Flexible & extensible focal point for optimization
Apache Arrow + Mnemonic

Extend Arrow In-Memory to In-Place using NVDIMM and 3DX SSD

Mnemonic

Collections
- Linked Lists
- Array
- Map
- Set

Core
- Mnemonic Non-Volatile Object Model
- Distributed Object Graph
- Native Computing

Allocator services
- Arrow Allocator
- nvml-vmem-service
- pmalloc-service

Arrow
- Vector Collection
- ArrowBuf

- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (e.g., Parquet-to-Arrow reader)
Spark + Mnemonic
Spark Experiment

- Non-Volatile RDD code is added into Spark, to alter data flow from Spark RDD via JVM heap to Mnemonic.
- Spill RDDs to off-heap and disk can be avoid.
- RDD data cache and local checkpoints can also be avoid when using nonvolatile objects.
Internal Structured Data Representation
(Example for 2D Linked List)

N nodes: represent the first level of linked list.

V nodes: represent the values of first level of linked list that still is a linked list

Only N2 and V1 (methods & volatile fields) are instantiated on heap
Apache Spark Mllib KMeans

```scala
val slotid = 10

val gftypes = Array(GenericField.GType.DURABLE, GenericField.GType.DOUBLE)
val proxy_elem = new EntityFactoryProxy() with Serializable {
    var val_efproxies:Array[EntityFactoryProxy] = null
    var val_gftypes:Array[GenericField.GType] = null
    if (null != factoryproxys && factoryproxys.length >= 2) {
      val_efproxies = Arrays.copyOfRange(factoryproxys, 1, factoryproxys.length);
    }
    if (null != gfields && gfields.length >= 2) {
      val_gftypes = Arrays.copyOfRange(gfields, 1, gfields.length);
    }
    NonVolatileNodeValueFactory.restore(allocator, val_efproxies, val_gftypes, phandler, autoreclaim)
  }
}
val efproxies:Array[EntityFactoryProxy] = Array(proxy_elem)

val examples = sc.textFile(params.input).map { line =>
  Vectors.dense(line.split(' ').map(_.toDouble))
}.cache()

var pmds = sc.nonVolatileLinkedDS[NonVolatileNodeValue[Double]](params.input, gftypes, efproxies, slotid)
val examples = pmds.map{x=>Vectors.dense(x.to[Array])}.cache()
```
Spark MLlib Kmeans Performance Improvement

- SPARK ML Kmeans implements one of the most popular clustering algorithms
  - Exp1: spark.storage.memoryFraction 0.6
  - Exp2: spark.storage.memoryFraction 0.2

<table>
<thead>
<tr>
<th>Exp1: GC Sum of 20 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 number of YoungGC</td>
<td>1310</td>
<td>864</td>
<td>0.660</td>
</tr>
<tr>
<td>Total FullGC Pause Real (s)</td>
<td>129.36</td>
<td>59.52</td>
<td>0.460</td>
</tr>
<tr>
<td>Total YoungGC Pause Real (s)</td>
<td>225.98</td>
<td>158.79</td>
<td>0.703</td>
</tr>
<tr>
<td>Total GC STW pauses (s)</td>
<td>355.34</td>
<td>218.31</td>
<td>0.614</td>
</tr>
</tbody>
</table>
Configuration for Performance Measurement

- **Platform Software Configuration**
  - OS: CentOS 7
  - JVM: JDK8 update 60
  - ParallelOldGC is used for Spark ML throughput workload Kmeans
  - Spark1.5.0

- **Platform Hardware**
  - Haswell EP (72 cores, E5-2699 v3 @ 2.3GHz)
  - 256GB DDR4-2133 on-board Memory
  - Storage media: SATA SSD
    - Device Model: INTEL SSDSC2BB016T4
    - Serial Number: BTWD442601311P6HGN
    - User Capacity: 1.60 TB
    - SATA Version is: SATA 3.0, 6.0 Gb/s

- **Spark Kmeans Experiments Setting:**
  - System memory is reduced to 128G via static dummy ramfile occupancy
  - 20 executors with 4 GB JVM heap, 3 cores
  - Both of datasets are stored on SSD
  - Experiment 1
    - spark.storage.memoryFraction 0.6
    - 100,000,000 records with vector in 12 dimension (20 partitions)
  - Experiment 2
    - spark.storage.memoryFraction 0.2
    - 100,000,000 records with vector in 6 dimension (20 partitions)
Project Links

Join Mnemonic Community: dev@mnemonic.incubator.apache.org

Source Code: https://github.com/apache/incubator-mnemonic

JIRA Tracking: https://issues.apache.org/jira/browse/MNEMONIC

Website: http://mnemonic.incubator.apache.org/

Contact:
yanpingw@apache.org; garyw@apache.org
Disclaimers

Apache Mnemonic is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by Apache Incubator PMC. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

While Mnemonic is operating within the time period where the application is processing their durable data. Mnemonic guarantees the persistency once Mnemonic is closed gracefully.