Which Memory Abstraction for NVRAMs – A Perspective from the Sage2 project

Gregory Vaumourin, Christophe Laferrière, Philippe Couvéé, Sebastien Valat
Atos BDS R&D Data Management, France
Who is this guy?

- PhD at INRIA Bordeaux (France) in 2016 with Denis Bartou, about memory caches management for embedded architectures

- Post-Doc at University of Uppsala (Sweden) with David Black-Schaffer about NVM for Memory caches

- Now working at ATOS in the R&D SW Data Management Team with Philippe Couvée
  - Tools for profiling/analyzing I/Os
  - I/O SW Accelerators
  - Flash-based Burst Buffers
Outline of the presentation

I – A bit of context with Sage2

II – How we plan to use NVDIMMs
   – Overall semantics
   – uMMAP-IO description

III – One Optimization for Checkpoint optimization
I - One Storage System to rule them all

Extreme Computing
- Changing I/O needs
- HDDs cannot keep up

AI/Deep Learning
- Large Memory Requirements
- I/O Requirements not well defined!

Big Data Analysis
- Avoid Data Movements
- Manage & Process extremely large data sets

SAGE
I- Sage2 Partners

"Sage2" Perципient StorAGe for Exascale data Centric computing

FETHPC Project ~€4M 9 partners
Sept 2018 – Aug 2021
I – MERO Introduction

- Mero: distributed object storage platform developed by Seagate
  - Tackle scalability problems of traditional file system for exascale
  - Handle a pool of nodes that might have non-volatile memory devices
  - Optimizations such as function shipping, object containers, ...
  - Will become open-source soon

- Clovis: API built on top of Mero:
  - Handle system conf, services management
I – Sage2 Storage Platform

- Archival Grade Storage (Tier-4)
- High Performance Scratch Storage (Tier-3)
- Flash Tier (Tier-2)
- NVRAM Tier (Tier-1)
- Node Local NVRAM (Tier-0)

AI workflow

Simulation with Data Analysis Pipeline

ARM processing environment

Compute Capability close to Tiers
I – Sage2 Software stack

Use Cases and Workflow

Tools (eg: Tensorflow on Objects)

Prog. Models (Transactional Object API)

Scheduling (eg: Resource and Data Aware)

Global Memory Abstraction

Object Storage Core
Mero + Clovis

Global Memory Abstraction

ARM Compute Platform
Node Local NVM Devices
I – Sage2 Objectives

▶ WP2: Design and implement software infrastructure needed for the Global Memory Abstraction on persistent storage tiers

▶ “The work package also delivers the necessary studies and support for accommodating and managing data with a new generation of Non Volatile Memories both within the compute node and externally.”

▶ Tasks:
  – First year: Global memory abstraction support design
  – Second year: Global memory abstraction support implementation
Outline

I – A bit of context with Sage2
II – How we plan to use NVDIMMs
   - Overall semantics
   - uMMAP-IO description
III – One Optimization for Checkpoint optimization
II – Mapping of MERO Objects

» Memory map » Mero objects from lower tiers:
  - Allow access to remote data
  - Mapping directly in the Memory/NVRAM of the local compute
  - Direct load/store interface can be used
 Complexity of interface between all tiers ...

 How to have a transparent interface for applications?

<table>
<thead>
<tr>
<th>Buffer-based</th>
<th>mapped object (load/store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj = clovis_obj_init(&amp;id, ...); ... clovis_obj_op(&amp;obj, CLOVIS_READ, ...); ... clovis_obj_op(&amp;obj, CLOVIS_WRITE, ...);</td>
<td>obj = clovis_obj_init(&amp;id, ...); ... base = mmap(&amp;obj, O_WRITE, size, ...); strcpy(base, “hello there”); ... msync(base, size, 0);</td>
</tr>
</tbody>
</table>

I/O happens on read/write calls

I/O happens on:
- Read/Write page faults
- Flush to persistence
II – uMMAP-IO Presentation

- uMMAP-IO\(^1\): memory mapping in user-space instead
- Allow parameters configuration:
  - Segment Size
  - Synchronization mechanism with storage
  - Eviction policy of memory segments

Similar interface as standard `mmap`

```c
int ummap(size_t size, size_t seg_size, int prot, int fd, off_t offset, int flush_interval,
int umsync(void *addr, int evict);
int umremap(void *old_addr, size_t size, int fd, off_t offset, int sync, void **new_addr);
int umunmap(void *addr, int sync);
```

\(^1\)uMMAP-IO: User-level Memory-mapped I/O for HPC, S.R. Gomez et Al. HiPC 2019

https://github.com/sergiorg-kth/ummap-io
Reserved Virtual Memory Address Space

- Segment #1: 0x1000000 … 0x1000999
- Segment #2: 0x1001000 … 0x1001999
- Segment #3: 0x1002000 … 0x1002999
- ... (…)

Mero Object

Sage2 Multi-tier Storage System
- NVRAM (Compute Node)
- NVRAM (Tiers 1)
- NVMe SSD
- High-Performance Scratch Storage
- Archival Storage
II – 1st Scenario: Mapping NVRAM from Tiers 1

Global Memory Abstraction

Direct Load/Store

Compute Node

MERO Service

Memory Mapping of Mero Object

Memory

NVRAM-C

Tiers 1 NVRAM

MERO Service

Tiers 2 Flash Disk

MERO Service
II – 2nd Scenario: Mapping NVRAM from Tiers 0

Global Memory Abstraction

Compute Node

Direct Load/Store

Memory

NVRAM

Ephemeral Regions Part of Global Address Space

Compute Node

MERO Service

Tiers 1 NVRAM

Tiers 2 Flash Disk

MERO Service

MERO Service
II – 3rd Scenario: Mapping from Tiers 2

Global Memory Abstraction

Compute Node

Direct Load/Store

Memory

Memory Mapping of Mero Object

NVRAM-C

MERO Service

Tiers 1 NVRAM

MERO Service

Tiers 2 Flash Disk

MERO Service
II – NVDRAM Integration in Sage2

1. `ummap(object_id)`

2. Init RDMA connexions
   - Send `ummap` request

3. Recover object from MERO

4. Copy MERO object in NVDIMM

5. Return pointer to higher layers

- **Application**
- **ummap library**
- **uMMAP-IO Driver Client**
- **uMMAP-IO Driver Server**
- **NVDIMM Driver**
- **NVRAM**
- **Clovis API**
- **MERO Server**

**Tier Distribution**
- Tiers 1
- Tiers 2
- Tiers 3

-component yet to be developed
-storage

---

18 | 26-09-2019 | Vaumourin Gregory | © Atos
II – Typical Scenario

- BOUT++: Simulation of Plasma
  - I/O bound, meshes can be in the order of TBs
  - Update one element at a time (~100 bytes)

We can directly map the whole mesh in NVRAM and access it seamlessly!
Outline

- I – A bit of context with Sage2
- II – How we plan to use NVDIMMs
  - Overall semantics
  - uMMAP-IO description
- III – One Optimization for Checkpoint optimization
III – Checkpoint Problem

**Checkpoint restart Problem:**
- Tend to dump too much data
- Complexity of full automation, no unique standard way to do it

Lots of pressure on storage system

Require twice as much memory
III – Object fork extension

- New semantic: uMMAP-IO_fork()
  - clones the remote object
  - Reattaches the new one through memory mapping
  - Similar to copy on write
- Flush only send the modified segment to lower Tier

```c
obj = obj_create()
mesh = obj_uMMAP-IO (obj)
compute()
obj_uMMAP-IO _fork(mesh)
obj_uMMAP-IO _flush(mesh)
compute()
...
```
Conclusion

► Our approach: Mapping memory addresses to Storage devices including NVRAM memory
  – With ummap-IO, we get lots of flexibility

► Opportunity to build new semantic that could help developers with their applications

► Entering the implementation phase:
  – Which PDMK library to start with?