Remote Persistent Memory

Progress Report and Recent Findings

Paul Grun
Cray, Inc.
Chair, OpenFabrics Alliance
• SNIA and the OpenFabrics Alliance (OFA) are working to enable and accelerate the adoption of Remote Persistent Memory…
  • By providing standards and enabling software
  • By providing open source APIs
• Which depends on a clear elaboration of ‘use cases’ for RPM
• It also depends on understanding the relevant characteristics of the underlying technology
  • Which we will be exploring today
SNIA & OFA Collaboration Process

1. OFA + SNIA
   • Develop a set of example use cases for RPM
   • Generate a Whitepaper outlining those use cases

2. SNIA NVMP TWG
   • Generate Whitepapers describing each use case

3. OFA OFIWG
   • Generate open source network APIs to support those use cases

Describe Requirements

List Key Use Cases  Define Network APIs
Remote PM (RPM) System & Memory Models

Organized into pools, accessed as memory

Can be configured as a flat address space, or as object storage. Or both.

Much Different from “Local” Persistent Memory
3 Use Cases for RPM have been Described

**USE CASE: HIGH AVAILABILITY, REPLICATION**
- Usage: Replicate data that is stored in local PM across a fabric and store it in remote PM.
- How it works:
  - What it looks like:
    - "High Availability".

**USE CASE: REMOTE PERSISTENT MEMORY**
- Usage: Expand on-node memory capacity, while taking advantage of persistence (or not). Disaggregates memory from compute.
- How it works:
  - What it looks like:
    - "Scalable Memory".

**USE CASE: SHARED PERSISTENT MEMORY**
- Usage: Information is shared among the elements of a distributed application. Persistence can be used to guard against node failure.
- How it works:
  - What it looks like:
    - "Scale-out Applications".
Selecting the right technology depends on understanding (at least):

**Key system design objectives**
- Scalability? In which dimension? Single server? Cluster?

**Application requirements**
- Is data being shared among threads or nodes?
- Are there application performance or capacity requirements?

Eventual API proposal should reflect a combination of *Use Cases* and *App Requirements*
To craft a network solution, and particularly to optimize the network software stack, there are number of factors to consider:

- **Consumer considerations**
  - For what purpose is the consumer storing/accessing persistent data remotely?
  - Under what conditions are data shared?
  - What is the security model?
- **System objectives**
  - For any given system, what are its design objectives? Performance? Scalability? High Availability?
  - What type of service is being offered? Object store? Pools of Memory?

The groundwork requires an understanding of:

- Application usage models
- Application requirements
- System Objectives
Which Boils Down to some Pretty Clear Solution Requirements

- **Data Availability/Protection**
  - Replicate local cache to RPM to achieve data availability

- **Local System Performance**
  - Eliminate disk accesses e.g. to stored databases

- **Scale Out Architectures**
  - Distributed databases, analytics applications, HPC parallel apps

- **Scale Up Architectures**
  - In-memory databases exceeding local DRAM capacity

- **Disaggregated Systems**
  - Compute capacity scales independently of memory capacity

- **Shared Data**
  - Simultaneous data access from multiple processes
  - Central shared repository for team collaboration on a large artifact

- **Improved Uptime, Fast Restart**
  - Quicker server recovery following power cycle
  - Checkpoint restart

- **Improved Disk Storage Performance**

---

**Question:** Which characteristics of RPM are relevant?

- First developed at an OpenFabrics Alliance Think Tank in 2017,
- Revised and extended at FMS 2018, and again at PM Summit 2019
It’s clear that Persistent Memory isn’t exactly memory, and it’s not precisely storage…

…so what is it?
How do we characterize it?
What role does it fill, exactly?
How do we use it to deliver the solutions described above?

* With thanks to SNL, 1/10/76
The Familiar Memory Hierarchy…

…with a wrinkle

Turns out that this new layer isn’t monolithic…

…and there are tradeoffs within the sublayers

Key characteristics:
Locality, Performance, Cost, Capacity
Introducing the ‘Characteristics’ Variable

• Many view the emerging Persistent Memory layer in the memory hierarchy as monolithic, evolving toward Nirvana
  • Nirvana defined as “infinite capacity, infinite bandwidth, zero latency, zero cost”
  • Oh, and “infinite retention”

• The truth is that there will always be tradeoffs
  • Performance vs Capacity vs Cost
  • Local vs Remote

• How to choose the right tradeoffs?

Assertion – understanding these characteristics, and how they map onto application use cases, will guide the development of networks to support RPM. Which is our ultimate goal.

Objective for today: Take a refined look at how the characteristics of RPM map onto solution requirements
## Example Target Apps for PM

<table>
<thead>
<tr>
<th>Database Applications</th>
<th>Data Analytics</th>
<th>Graph Analytics</th>
<th>Commercial Applications</th>
<th>HPC Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A modifiable, in-memory database that survives power cycles</td>
<td>• Create a database once, run new queries repeatedly</td>
<td>• Operate on larger graphs than would fit in local memory</td>
<td>• Enable collaboration on large scale projects</td>
<td>• Scalability, parallel applications • Checkpointing</td>
</tr>
</tbody>
</table>

**Database Applications**
- Persistence, capacity

**Data Analytics**
- Capacity, density, performance

**Graph Analytics**
- Persistence, capacity, cost
Persistence? Not Always Required

- **Persistence is valuable for:**
  - High Availability applications
  - Reducing or eliminating the need to access slower media, e.g. HDDs
  - Data protection and preservation

- **Persistence not required, but nice to have:**
  - Certain applications, such as analytics, that require establishing a database. Build the database once, run multiple queries against it
  - Collaborative workspaces

- **Other characteristics may prove to be more valuable than persistence**

If the app doesn’t need persistence, then the so-called convergence of storage and memory is uninteresting
There are Always Tradeoffs. For Example…

- **Performance**
  - Persistence may come at the cost of performance (but not always)

- **Cost**
  - If you can accept a lower level of performance, or you do not care much about access models (e.g. file/block/byte addressability), there may be lower cost options available

- **Capacity**
  - To achieve higher capacity, you might sacrifice persistence, performance, or cost
1st Order Tradeoff: Locality

- Some requirements are met by siting persistent memory on the compute node
  - Capacity-based applications
  - Some data protection usages
  - Replacement of local storage for performance reasons

- Others are only achieved by distributing persistent memory
  - Compute/memory disaggregation
    - independent scaling of compute and memory
  - Shared resource / shared data
  - Team collaboration
  - Distributed/Scale-out applications

Needless to say, this is our focus at the moment - RPM

Local may be synchronous, Remote is almost certain to be asynchronous
Local Persistent Memory

- **Data Availability/Protection**
  - Replicate local cache to RPM to achieve high availability

- **Local System Performance**
  - Eliminate disk accesses

- **Scale Out Architectures**
  - Scale out distributed databases, analytics applications, HPC parallel applications

- **Scale Up Architectures**
  - Scale up databases that exceed local memory capacity

- **Disaggregated System Architectures**
  - Compute capacity scales independently of memory capacity

- **Shared Data**
  - Support simultaneous data access to large teams

- **Improved Uptime, Fast Restart**
  - Quick server recovery following power cycle
  - Checkpoint restart

<table>
<thead>
<tr>
<th>Persistence</th>
<th>Performance</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓✓✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓✓✓</td>
<td>✓✓✓</td>
</tr>
<tr>
<td>✓✓✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Remote PM

- **Data Availability/Protection**
  - Replicate local cache to RPM to achieve data availability

- **Local System Performance**
  - Eliminate disk accesses e.g. to stored databases

- **Scale Out Architectures**
  - Distributed databases, analytics applications, HPC parallel apps

- **Scale Up Architectures**
  - In-memory databases exceeding local DRAM capacity

- **Disaggregated Systems**
  - Compute capacity scales independently of memory capacity

- **Shared Data**
  - Simultaneous data access from multiple processes
  - Central shared repository for team collaboration on a large artifact

- **Improved Uptime, Fast Restart**
  - Quicker server recovery following power cycle
  - Checkpoint restart
Data Availability, Protection

What it looks like

<table>
<thead>
<tr>
<th></th>
<th>Persistence</th>
<th>Performance</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>√√√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Checkpoint</td>
<td>√√√</td>
<td>√√</td>
<td>√</td>
</tr>
</tbody>
</table>

How it works

Data Availability/Protection
Usage: Replicate local cache to RPM to achieve data availability
Scale Out

Scale Out Architectures
Usage: Distributed databases, analytics applications, HPC parallel apps

Disaggregated Systems
Usage: Compute capacity and memory capacity scale independently

<table>
<thead>
<tr>
<th></th>
<th>Persistence</th>
<th>Performance</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Out</td>
<td>✓</td>
<td>★★★</td>
<td>★</td>
</tr>
<tr>
<td>Disaggregation</td>
<td>✓</td>
<td>★★★</td>
<td>★</td>
</tr>
</tbody>
</table>

“Scalable Memory”
Shared Data

Usage: Simultaneous data access from multiple processes
Usage: Central shared repository for team collaboration

Remote Shared Memory Service

<table>
<thead>
<tr>
<th></th>
<th>Persistence</th>
<th>Performance</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Data</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Call to Action

- Key use cases have been described
- Application requirements and system objectives outlined
- Varying characteristics of RPM mapped onto the above
- Call to Action: Engage with either/both
  - OFA’s OpenFabrics Interfaces WG to develop the above,
    - www.openfabrics.org → Organization → OFA Calendar
  - SNIA’s NVMP TWG to develop the detailed use case Whitepapers