Persistent Memory over Fabrics
An Application-centric view

Paul Grun
Cray, Inc
OpenFabrics Alliance Vice Chair
Agenda

OpenFabrics Alliance Intro

OpenFabrics Software

Introducing OFI - the OpenFabrics Interfaces Project

OFI Framework Overview – Framework, Providers

Delving into Data Storage / Data Access

Three Use Cases

A look at Persistent Memory
The OpenFabrics Alliance (OFA) is an open source-based organization that develops, tests, licenses, supports and distributes OpenFabrics Software (OFS). The Alliance’s mission is to develop and promote software that enables maximum application efficiency by delivering wire-speed messaging, ultra-low latencies and maximum bandwidth directly to applications with minimal CPU overhead.

https://openfabrics.org/index.php/organization.html
OpenFabrics Alliance – selected statistics

- Founded in 2004

- Leadership
  - Susan Coulter, LANL – Chair
  - Paul Grun, Cray Inc. – Vice Chair
  - Bill Lee, Mellanox Inc. – Treasurer
  - Chris Beggio, Sandia – Secretary (acting)
  - 14 active Directors/Promoters (Intel, IBM, HPE, NetApp, Oracle, Unisys, Nat'l Labs…)

- Major Activities
  1. Develop and support open source network stacks for high performance networking
     - OpenFabrics Software - OFS
  2. Interoperability program (in concert with the University of New Hampshire InterOperability Lab)
  3. Annual Workshop – March 27-31, Austin TX

- Technical Working Groups
  - OFIWG, DS/DA, EWG, OFVWG, IWG
  - https://openfabrics.org/index.php/working-groups.html (archives, presentations, all publicly available)

 today’s focus
OpenFabrics Software (OFS)

Open Source APIs and software for advanced networks

Emerged along with the nascent InfiniBand industry in 2004

Soon thereafter expanded to include other IB-based networks such as RoCE and iWARP

Wildly successful, to the point that RDMA technology is now being integrated upstream. Clearly, people like RDMA.

OFED distribution and support: managed by the Enterprise Working Group – EWG

Verbs development: managed by the OpenFabrics Verbs Working Group - OFVWG
Historically, network APIs have been developed ad hoc as part of the development of a new network.

To wit: today’s Verbs API is the implementation of the verbs semantics specified in the InfiniBand Architecture.

But what if a network API was developed that catered specifically to the needs of its consumers, and the network beneath it was allowed to develop organically?

What would be the consumer requirements? What would such a resulting API look like?
Introducing the OpenFabrics Interfaces Project

- OpenFabric Interfaces Project (OFI)
  - Proposed jointly by Cray and Intel, August 2013
  - Chartered by the OpenFabrics Alliance, w/ Cray and Intel as co-chairs
- Objectives
  - ‘Transport neutral’ network APIs
  - ‘Application centric’, driven by application requirements

“Transport Neutral, Application-Centric”

OFI Charter

Develop, test, and distribute:

1. An extensible, open source framework that provides access to high-performance fabric interfaces and services
2. Extensible, open source interfaces aligned with ULP and application needs for high-performance fabric services

**OFIWG will not create specifications**, but will work with standards bodies to create interoperability as needed
OpenFabrics Software (OFS)

Result:
1. An extensible interface driven by application requirements
2. Support for multiple fabrics
3. Exposes an interface written in the language of the application
OFI Framework

OFI consists of two major components:
- a set of defined APIs (and some functions) – libfabric, kfabric
- a set of wire-specific ‘providers’ - ‘OFI providers’

Think of the ‘providers’ as an implementation of the API on a given wire

a series of interfaces to access fabric services

Provider exports the services defined by the API
All providers expose the same interfaces, None of them expose details of the underlying fabric.

Current providers: sockets, verbs, usnic, gni, mlx, psm, psm2, udp, bgq
OFI Framework – a bit more detail

Network consumers

**Framework**
- Control Services
  - Discovery
- Communication Services
  - Connection Management
  - Address Vectors
- Completion Services
  - Event & Completion Queues
  - Counters
- Data Transfer Services
  - Msg queues
  - RMA
  - Tag Matching
  - Atomics

**Provider**
- Discovery
- Connection Management
- Address Vectors
- Event & Completion Queues
- Counters
- Msg queues
- RMA
- Tag Matching
- Atomics

**NIC**
Application-centric design means that the working groups are driven by *use cases*:

Data Storage / Data Access, Distributed and Parallel Computing…
OFI Project Overview – work group structure

OFI Project

Legacy Apps
- Sockets apps
- IP apps

Data Analysis
- Structured data
- Unstructured data

Data Storage, Data Access
- Data Storage
  - object, file, block
  - storage class memory
- Data Access
  - remote persistent memory

Distributed and Parallel Computing
- Msg Passing
  - MPI applications
- Shared memory
  - PGAS languages

Kernel mode
‘kfabric’

User mode
‘libfabric’

(Not quite right because you can imagine user mode storage, e.g. CEPH)
OFI Project Overview – work group structure

Since the topic today is Persistent Memory…
Data Storage, Data Access?

**Data Storage / Data Access**

- **DS**
  - object, file, block
  - storage class mem

- **DA**
  - persistent memory

**Reminder:**
libfabric: User mode library
kfabric : Kernel modules

remote storage class memory, persistent memory
At what point does remote storage begin to look instead like remote persistent memory? How would applications treat a pool of remote persistent memory?

Key Use Cases:
1. Lustre
2. NVMe
3. Persistent Memory
Backport LNET to kfabric, and we’re done. Any kfabric provider after that will work with Lustre.
Data Storage Example – NVMe
Data Storage – enhanced APIs for storage

Places where the OFA can help:
- kfabric as a second native API for NVMe
- kfabric as a possible ‘LND’ for LNET
Applications tolerate long delays for storage, but assume very low latency for memory
- Storage systems are generally asynchronous, target driven, optimized for cost
- Memory systems are synchronous, and highly optimized to deliver the lowest imaginable latency with no CPU stalls

Persistent Memory over fabrics is somewhere in between:
- Much faster than storage, but not as fast as local memory

How to treat PM over fabrics?
- Build tremendously fast remote memory networks, or
- Find ways to hide the latency, making it look for the most part like memory, but cheaper and remote
Two interesting PM use cases

Consumers that handle remote memory naturally…

• SHMEM-based applications, PGAS languages…

Requirements:
• optimized completion semantics to indicate that data is globally visible,
• semantics to commit data to persistent memory
• completion semantics to indicate persistence

…and those that don’t (but we wish they did)

• HA use case

Requirements:
• exceedingly fast remote memory bus, or a way to hide latency
• the latter requires an ability to control when data is written to PM
PM high availability use case

user

file access

mem mapping

user

cache writes

store, store, store...

flush

persistent memory

load/store

user

kernel

memory controller

file system

libfabric

provider

kfabric

provider

NIC

persistent memory

write, write, write...

cache
For ‘normal’ fabrics, the responder returns an ACK when the data has been received by the end point.

- It may not be globally visible, and/or

- It may not yet be persistent

Need an efficient mechanism for indicating when the data is globally visible, and is persistent
Data Access – key fabric requirements

Objectives:
1. Client controls commits to persistent memory
2. Distinct indications of global visibility, persistence
3. Optimize protocols to avoid round trips

Caution: OFA does not define wire protocols, it only defines the semantics seen by the consumer
Possible converged I/O stack
Discussion – “Between Two Ferns”

Doug Voigt - SNIA NVM PM TWG Chair
Paul Grun – OFA Vice Chair, co-chair OFIWG, DS/DA

Are we going in the right direction?
Next steps?
Thank You
Backup
kfabric API – very similar to existing libfabric

**Consumer APIs**
- `kfi_getinfo()`  `kfi_fabric()`  `kfi_domain()`  `kfi_endpoint()`  `kfi_cq_open()`  `kfi_ep_bind()`
- `kfi_listen()`  `kfi_accept()`  `kfi_connect()`  `kfi_send()`  `kfi_recv()`  `kfi_read()`  `kfi_write()`
- `kfi_cq_read()`  `kfi_cq_sread()`  `kfi_eq_read()`  `kfi_eq_sread()`  `kfi_close()`  …

**Provider APIs**
- `kfi_provider_register()`
  During kfi provider module load a call to `kfi_provider_register()` supplies the kfi api with a dispatch vector for `kfi_*` calls.

- `kfi_provider_deregister()`
  During kfi provider module unload/cleanup `kfi_provider_deregister()` destroys the `kfi_*` runtime linkage for the specific provider (ref counted).

http://downloads.openfabrics.org/WorkGroups/ofiwg/dsda_kfabric_architecture/
NVMe/F is an extension, allowing access to an NVMe device over a fabric.

NVMe/F leverages the characteristics of verbs to good effect for verbs-based fabrics.