**OFI Data Storage / Data Access Subteam Weekly telecom – 07/19/2016**

**DS/DA Shared Documents:** <http://downloads.openfabrics.org/WorkGroups/ofiwg/>

**Agenda**

* roll call, agenda bashing

Persistent memory stuff

* Complete the discussion of Chet Douglas’ proposal for extending RDMA support for persistent memory access.
* Applicability to kfabric?

Update (if any) on kfabric GitHub repo – outstanding issues?

**Intel Proposal for Extending RDMA support for PM access – Chet Douglas**

**See the document “RDMA Extensions-Proposed libfabric API.DOCX**

* This is a follow-up to a presentation Chet gave to this group probably about a year ago on Intel’s thinking about using software methods to make persistent writes durable.
* Intel is looking at a number of s/w and h/w changes for its next platform. The previous mechanisms required a number of extra software steps to make it work.
* This is a libfabric proposal for a ring 3 API, and what extensions to it might look like, driven by an internal Intel hardware assessment of data paths and so on.
* One key objective: reduce h/w complexity. The proposal recognizes that the h/w design pipeline can be long.
* Chet will publish a new version of the doc following this meeting.
* Q: are these enhancements specific to OFI, or can they be extended to verbs? A: Should mirror nicely into verbs.
* Proposing changes to three libfabric APIs: fi\_getinfo, fi\_mr\_reg, fi\_writemsg
* **fi\_getinfo** – changes to the info flags to indicate the existence of persistent memory, i.e. this device supports accesses on something other than a block granularity.
* **fi\_mr\_reg** – proposing three new flags: fi\_pmem to indicate that the memory region being registered is persistent. fi\_uncached is a hint to indicate to the provider that this region should not be cached. Helps the NIC decide how to handle caching for this memory region. This hint applies to the target side; it is an open question as to whether it should be made available on the initiator side. fi\_non\_standard\_memory\_device, mainly for use if the PM is not attached to a memory bus. Allows kernel driver to supply whatever resources the NIC may need. Q: Current libfabric doesn’t distinguish between L\_Key and R\_Key, today you get a descriptor (equiv to an L\_Key) and a key (equivalent to an R\_Key). The group seemed to agree that this flag should not be exposed across the API to the consumer, but is useful to the provider implementation.
* **fi\_writemsg** – asking for new op codes for fi\_write\_commit, and fi\_write\_commit\_immediate. For the moment, include these as flags to the existing fi\_writemsg API. The provider will probably use these flags to create a new op code. New flags: FI\_COMMIT basically gives the completion semantics of an RDMA READ, i.e., you get a completion when data in scope has reached the global visibility point.

Resuming on 6/21/16

* There is a corresponding Intel protocol proposal to go along with this.
* **fi\_writemsg** – see above. Additional flags to the API, which is unchanged. New flags: (continued from above) FI\_COMMIT modifies the normal completion semantic on the initiator side, probably results in a new opcode on the wire. FI\_COMMIT is specific to a particular connection and R\_key. (The expression “QP” should be changed to endpoint, which is the semantics used by libfabric.) For non-volatile memory, FI\_COMMIT indicates that the data has reached the global visibility point, but is not durable. FI\_IMMED modifies the completion semantics on the target side. This is different from a write with immediate data, and may need a different name for the flag. If it doesn’t include some sort of immediate data, need to figure out how to signal the context to the target. FI\_FENCE causes a fence on the TARGET SIDE, which guarantees that previous writes with the same R\_KEY will be made durable before executing the write fenced with the same R\_KEY. There is some question as to whether writes coming after the fence could pass the fenced write.
* **Ordering and Completion semantics**
* Any fi\_writes that don’t take the previous flags cannot be used for PMEM.
* Ordering applies only to operations on a given connection, there is no notion of ordering between connections, or between regions registered with different R\_keys. There are no ordering guarantees in the absence of the FI\_FENCE flag, even for writes on the same connection to the same R\_key region. The ordering semantics should remain identical to the existing semantic.
* Today, there is no mechanism for guaranteeing ordering between writes to two different regions. This may be a problem for the case e.g. where the first write writes a blob of data and a second write is used to update pointers, and they are in different memory regions.
* **Open issues**
* Ordering – does the FI\_FENCE impact subsequent writes or not? One point of view is that a fence marks a point in time, everything before the fence gets completed before any subsequent writes are written. Needs some work here.
* Should an FI\_FENCE be allowed to force ordering on a connection basis, irrespective of memory region?

Resuming on 7/19/16

* FI\_NON\_STANDARD\_MEMORY\_DEVICE flag will be removed.
* FI\_COMMIT will probably result in a new on-the-wire opcode. This is being presented to the IBTA LWG, which is just now beginning its work on NVM.
* **Open Issues Discussion (cont’d)**
* The question of the semantics of Fencing is still an open. Intel’s current ‘fence’ does not prevent subsequent writes from passing the fenced operation. The question is whether a fence is on a per endpoint/memory region basis or not. AR Chet Douglas – plans to come back and discuss the proposed “final” solution.
* Key question: Does the fence apply to all memory regions associated with a given EP, or is it on a per EP/memory region basis?
* Current assumption is that SQ, RQ and CQ (such as they are) are not in persistent memory. Mainly because it complicates the design if they are, although they could be if there is a compelling reason to do so.
* No good way at present to report what atomicity guarantees are provided.

Other efforts in the industry:

* Intel’s API proposal, Tom Talpey’s IETF Draft, SNIA HA White Paper
* Major difference is in the commit list. The Intel proposal is an implicit commit, the SNIA and IETF drafts both contain an explicit commit list.
* There is a corollary w.r.t. ordering – for an implicit commit list (Intel) the ordering is implied, for the SNIA proposal ordering is implied by optimized flush semantics. IETF Draft builds in a special (optional) 64 bit update that is written last (flag update).
* It boils down to flexibility vs the burden on the ULP – there is a non-zero burden on the consumer to maintain an ordered commit list.
* NetApp has already discussed this internally with Intel.

W.R.T. kfabric

- All these changes should also be included in kfabric.

Next steps

* Planning to submit a series of patches/pull requests to the existing libfabric API, but planning to wait to hear the feedback from the IBTA. However, caution is advised since the IBTA is likely to look at the issue primarily from the perspective of the wire, whereas OFI is looking at it mainly from the perspective of the consumer.

**Next Agenda – 8/2/16**

* NetApp and Cray to present discussion on the relative merits of the ordering question discussed above.
  + ‘barrier’ vs ‘fence’ – what are the impacts on subsequent write operations?
  + Ordering questions w.r.t. an explicit commit list. What are the required ordering semantics?

**Webex Recording:**

|  |  |
| --- | --- |
| [**Play recording**](https://cisco.webex.com/ciscosales/ldr.php?RCID=af61603e3dde9ad8424a285da11b3ff8) (50 min) | |
| Recording password: eNmemqM9 |  |

**Next regular telecom:**

Next meeting: Tuesday, 8/02/16

8am-9am Pacific daylight time

**NOTE:** We have switched over to using Webex (courtesy of Cisco). The URL for joining meetings is:

[Join WebEx meeting](https://cisco.webex.com/ciscosales/j.php?MTID=m221d8a20185d84b30daa0096aca0f182)

**Join by phone**

+1-866-432-9903 Call-in toll-free number (US/Canada)

+1-408-525-6800 Call-in toll number (US/Canada)

Access code: 201 212 241