OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

HPNL: A HIGH-PERFORMANCE LIGHT WEIGHTED NETWORK LIBRARY FOR BIGDATA APPLICATION

Haodong Tang, Jian Zhang, Fred Zhang

{haodong.tang, jian.zhang, fred.zhang} @intel.com

Intel

[March, 2019]

LEGAL NOTICE AND DISCLAIMERS

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel, the Intel logo, 3D Xpoint, Optane, Optane DCPMM are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

LEGAL INFORMATION: BENCHMARK AND PERFORMANCE DISCLAIMERS

- Performance results are based on testing as of Feb. 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information, see Performance Benchmark Test Disclosure.
- Configurations: see performance benchmark test configurations.

AGENDA

- Background and motivation
- RDMA enabling in bigdata software ecosystem
- Transportation agnostic messenger proposals – HPNL
- Optimizing Spark shuffle with HPNL and Persistent Memory
- Summary & Next steps

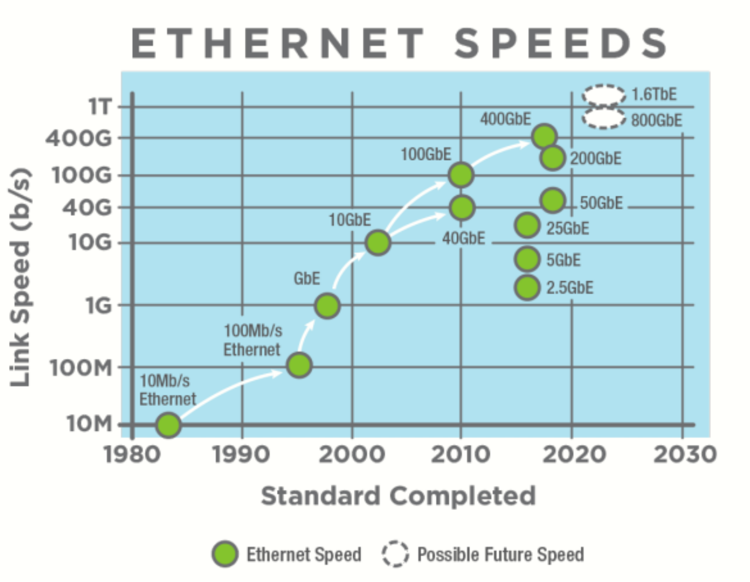
BACKGROUND AND MOTIVATION

BACKGROUND AND MOTIVATIONS

- **Bigdata analytics software stack requires RDMA for higher performance**
  - Spark is expected to achieve high throughput & ultra-low latency for different workloads like ad-hoc query, real-time streaming, and machine learning
  - There are bottlenecks to be improved in shuffle phase
  - While RDMA networking can leads to orders of magnitude improvement, using VERB interface is a challenge for application development and porting
- **Motivations**
  - A light-weight network library built on Libfabric
  - Protocol-independent networking framework that can easily run on all transportation protocols: TCP, RDMA, IB, OPA etc.
  - Flexible interfaces & abstractions: C/JAVA API and high-level abstraction to let developer easily replace other TCP/IP based network library, like ASIO or Netty

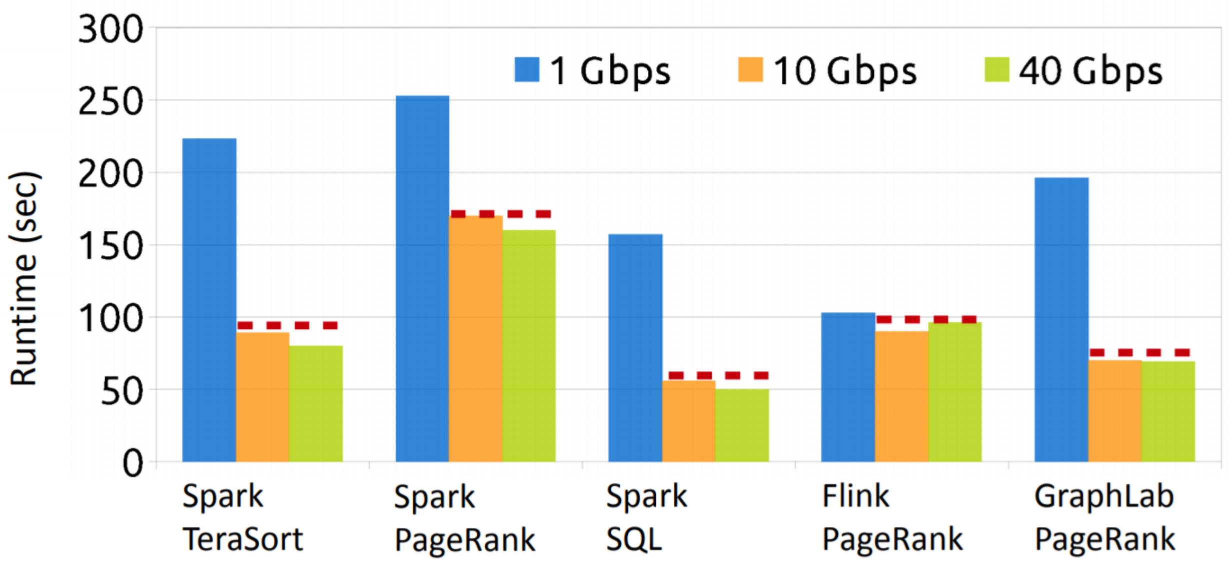
BIGDATA APPLICATION MEETS MODERN NETWORK TECHNOLOGY

- **Network interconnects have evolved**
  - Bandwidth from 1Gbps to 100Gbps
  - Message RTT latency reduced by an order of magnitude



Ethernet Speed *

- **BigData Application cannot benefit from new HW**
- **Network consecutive evolving doesn't result in application consecutive speedup.**



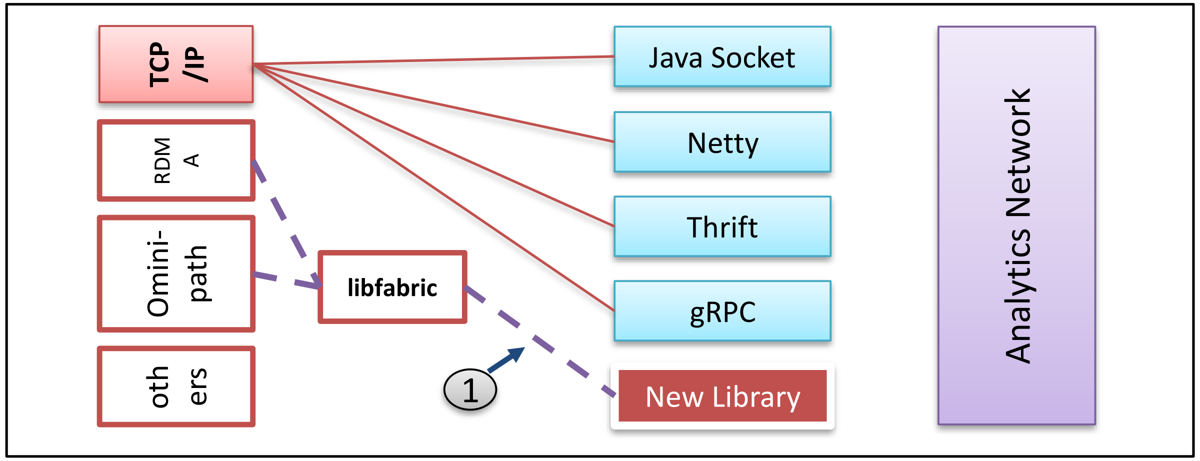
Bigdata workloads performance **

Source
*: https://ethernetalliance.org/the-2018-ethernet-roadmap
**: https://www.openfabrics.org/images/eventpresos/2017presentations/109_Crail_BMetzler.pdf

RDMA ENABLING IN BIGDATA ECOSYSTEM

APPROACH 1 – INTEGRATE LIBFABRIC INTO BIGDATA APP CASE BY CASE

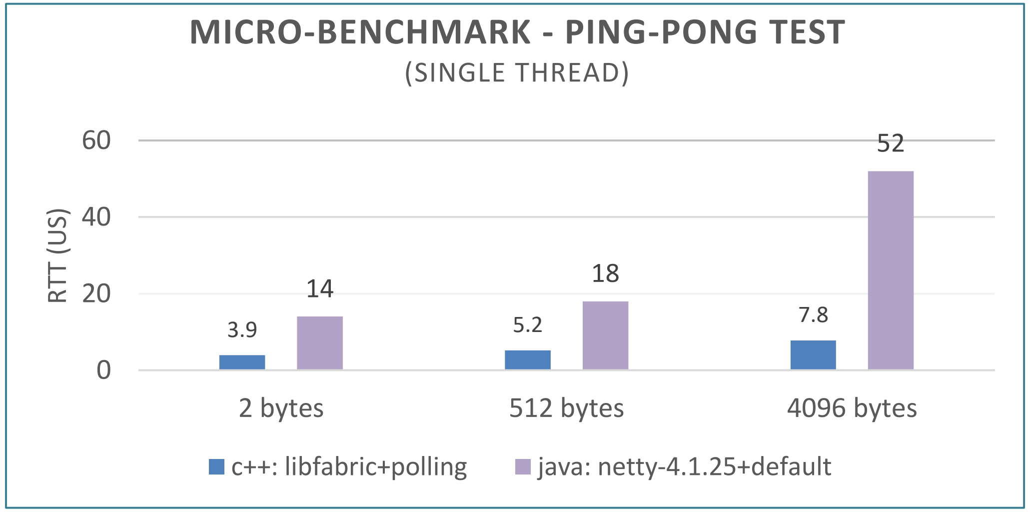
- **Existing projects for BigData w/ RDMA**
  - SparkRDMA.
    - TeraSort benchmark shows 2.63x overall reduced in execution time.*
  - HiDB project from OSU.
    - HiBench PageRank total time reduced by 37%-43% over IPoIB.**
- **Pros**
  - Easy to integrate to one BigData application
- **Cons**
  - Hard to benefit all the BigData App, need to integrate case by case.
  - Can't enable cross-stack optimization.



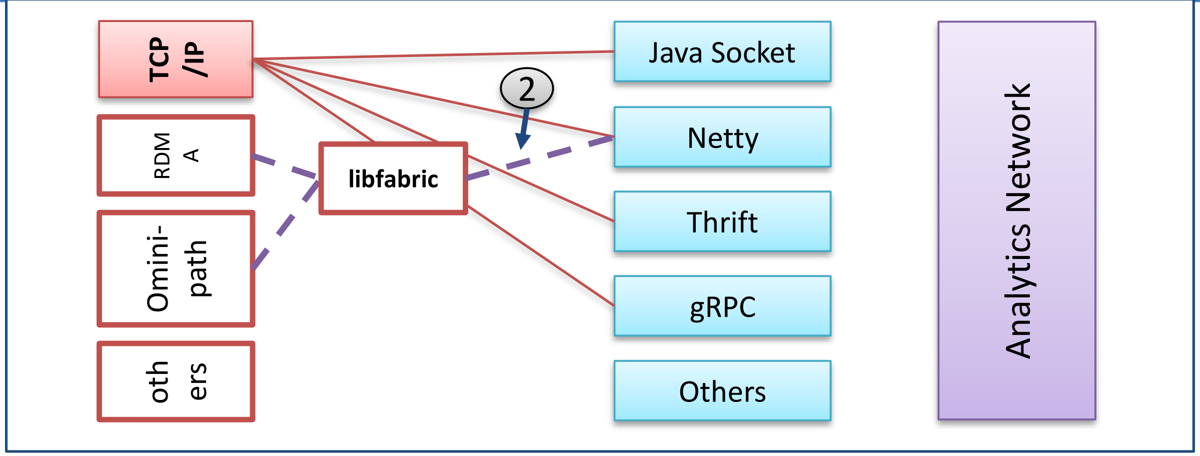
* https://github.com/Mellanox/SparkRDMA
** http://hibd.cse.ohio-state.edu/performance/pagerank/

APPROACH 2 – INTEGRATE LIBFABRIC INTO EXISTING LIB

- **Netty* is [widely](https://netty.io/wiki/adopters.html)used in BigData application.**
- **However, message transfer is more expensive than RDMA.**
  - Message RTT time is higher.
  - Consumes more CPU.



*   https://netty.io/index.html
** https://netty.io/wiki/adopters.html



Pros

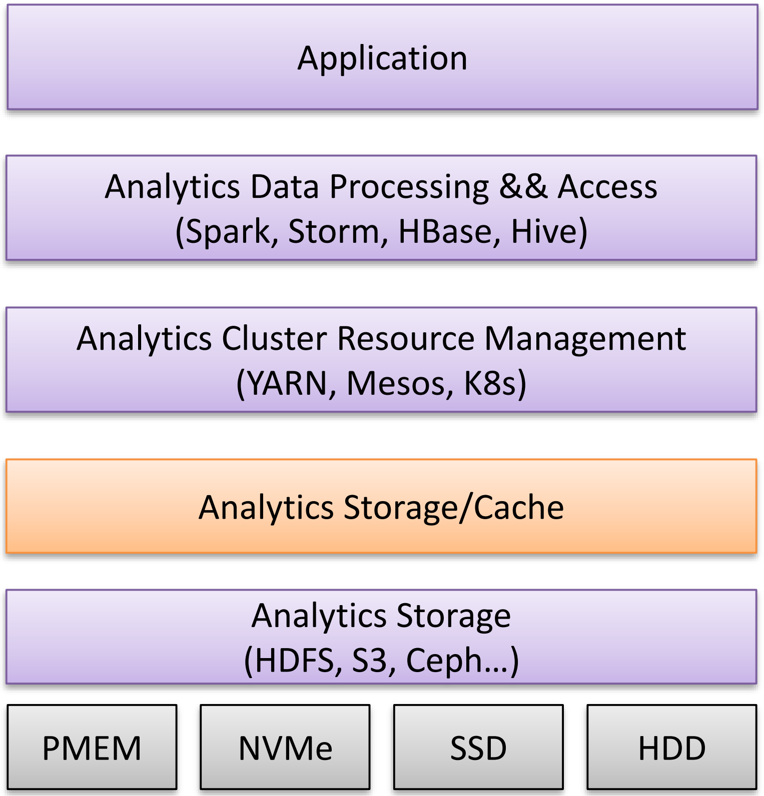
- When Netty supports RDMA & Omni-Path, all the BigData App based on Netty will benefit directly.

Cons

- Hard to implement all the interface in Netty transport layer.
- Can’t enable cross-stack optimization.

APPROACH 3 – FULL STACK OPTIMIZATION FOR DATA STORAGE

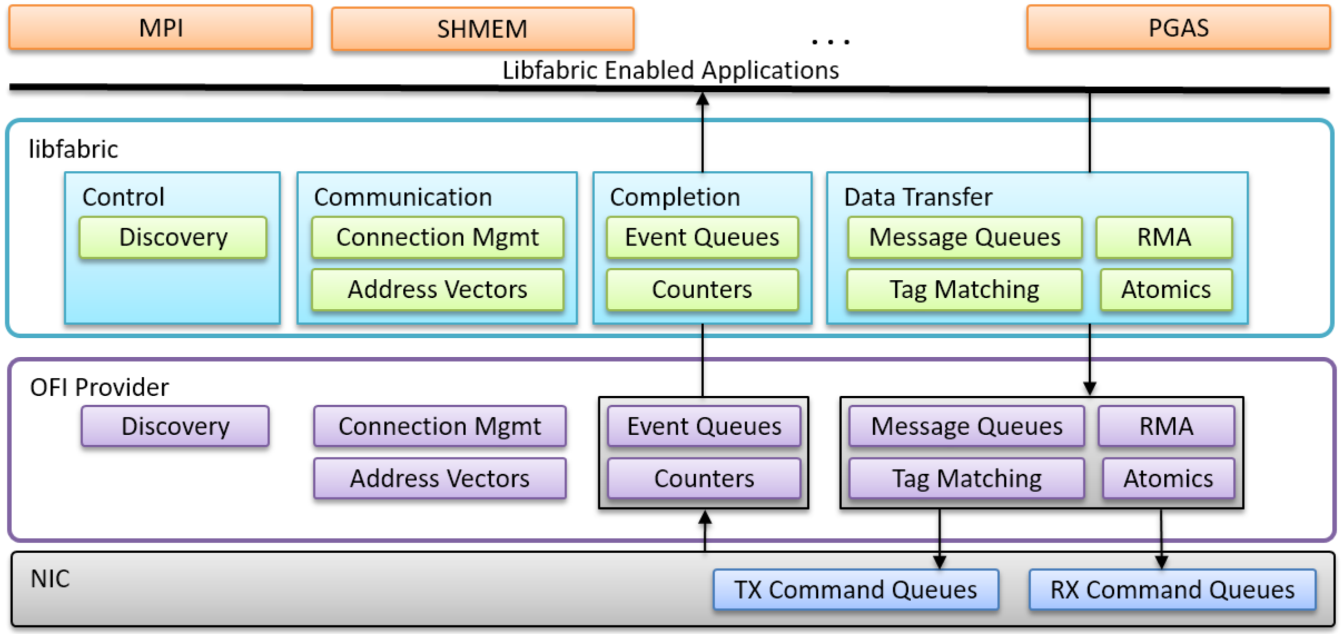
- **Existing projects for BigData by this way.**
  - Crail from IBM.
  - FlashNet: Flash/Network Stack Co-Design
  - Octopus: An RDMA-enabled Distributed Persistent Memory File System
- **Pros**
  - Co-designed storage/network stack optimized to reduce cross-stack overhead between network and flash IO.



TRANSPORTATION AGNOSTIC MESSENGER– HIGH PERFORMANCE NETWORK LIBRARY (HPNL)

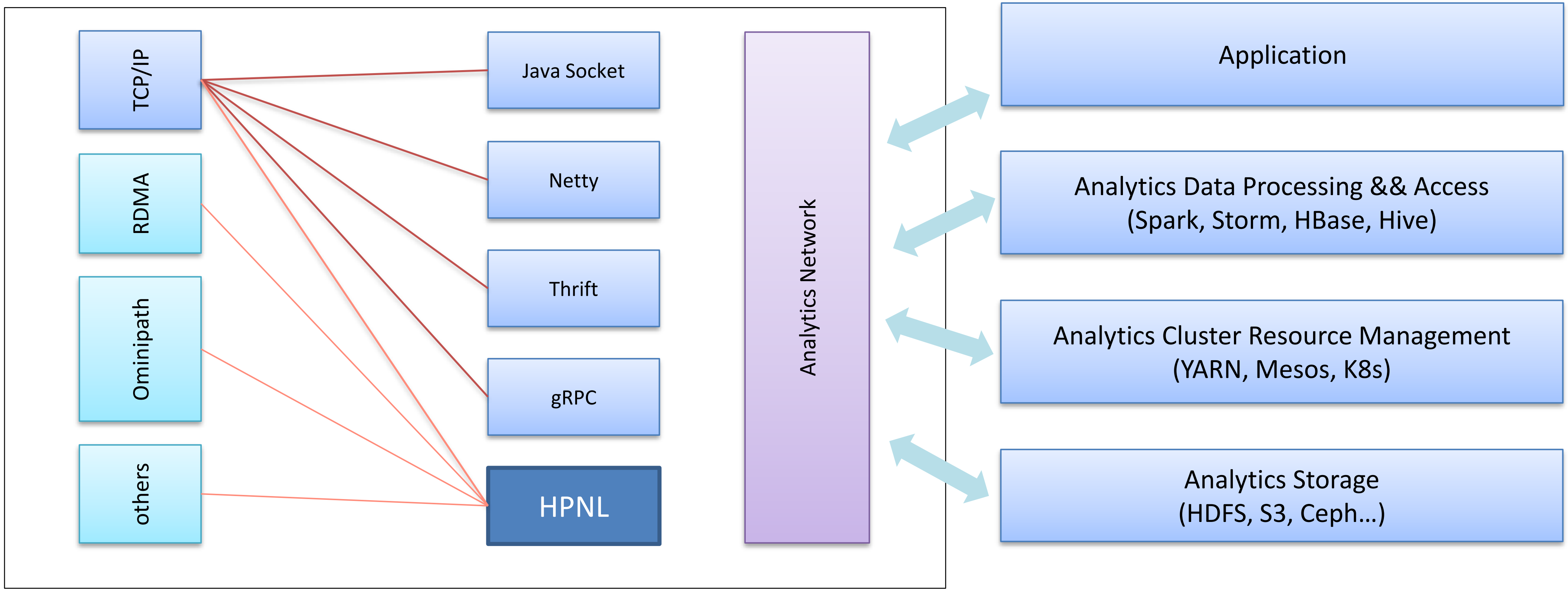
WHY LIBFABRIC?

- Libfabric is developed by the OFI Working Group, a subgroup of OFA.
- Goals: to define interfaces that enable a tight sematic map between application and underlying fabric services.
- **Libfabric is friendly to application developers, transportation protocol agnostic, easy to port and migrate to new hardware.**



*source: https://ofiwg.github.io/libfabric/

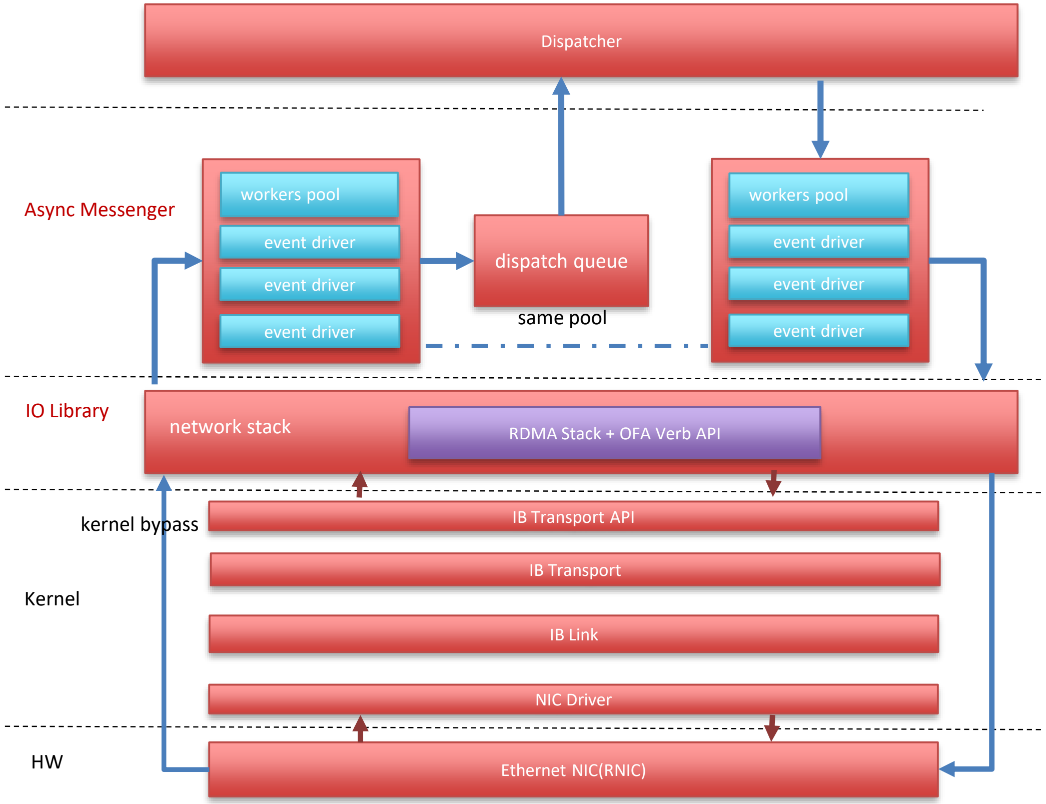
WHERE DOES HPNL FIT IN THE BIGDATA STACK?



- **HPNL is a general-purpose network library and a appropriated choice to accelerate BigData application with HPC network technology.**

PREVIOUS WORK: RDMA IN CEPH

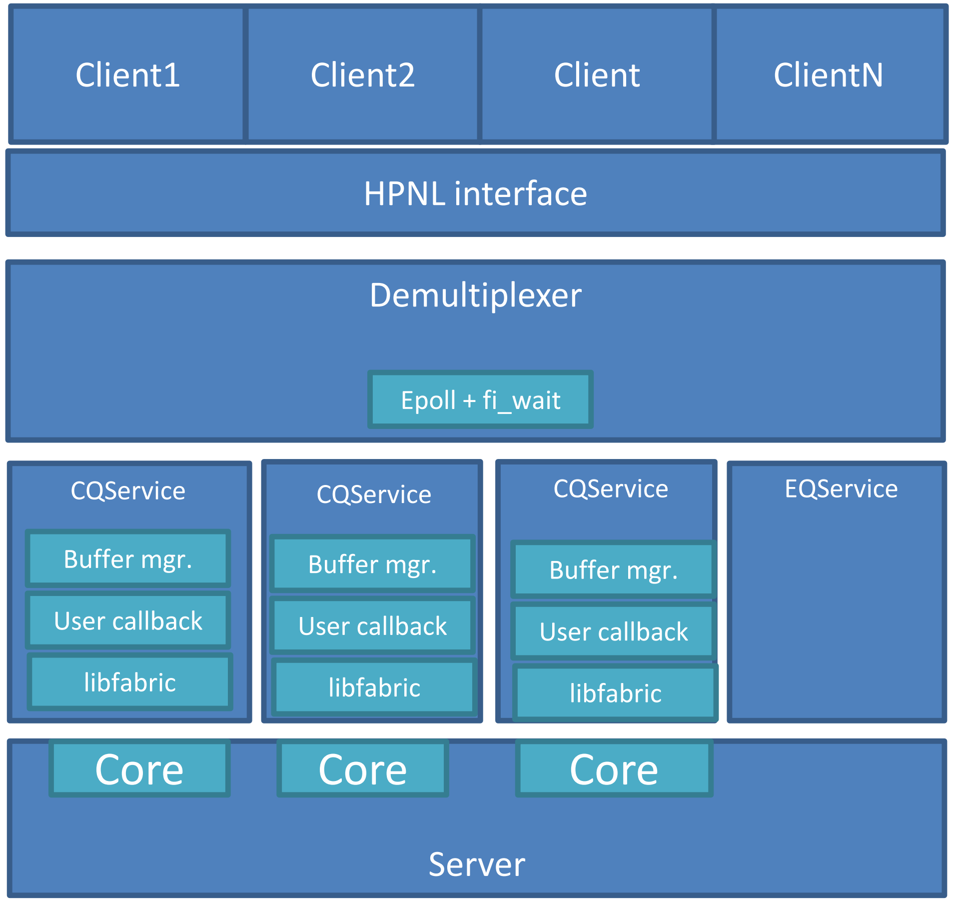
- **XIO Messenger.**
  - Based on Accelio, seamlessly supporting RDMA.
  - Scalability issue.
  - Merged to Ceph master three years ago, no support for now.
- **Async Messenger.**
  - Async Messenger is compatible with different network protocol, like Posix, RDMA and DPDK.
  - Current RDMA implementation supports IB protocol.
- **We implemented iwrap based RDMA for Ceph and pushed to upstream ***
  - Showed 17% performance improvement for 4K random write compared with TCP/IP
  - Connection management: RDMA-CM based RDMA connection management
  - Queue pairs: centralized memory pool for recv queue (RQ)
- **Extended the work, build a developer friendly library for bigdata applications**



* Accelerating Ceph with RDMA and NVMe-OF: OFA 2018

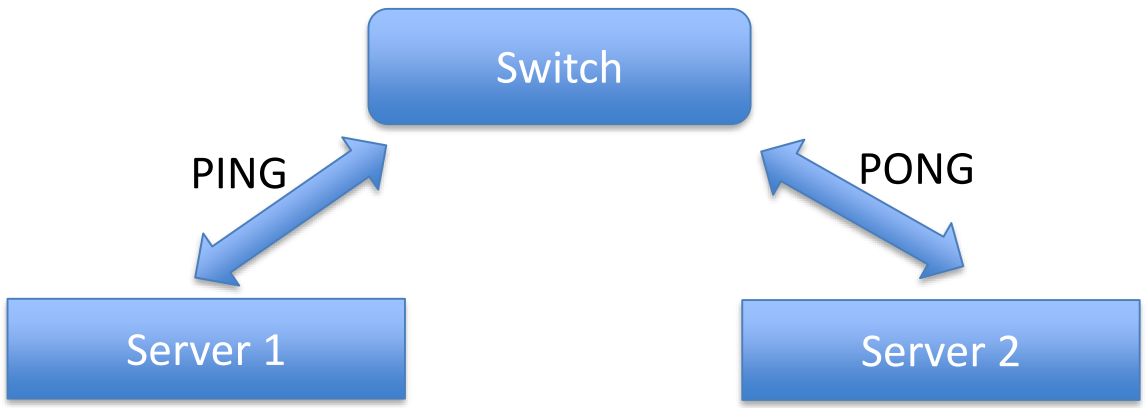
HPNL ARCHITECTURE

- **Zero-copy approach**
  - The HPNL buffer allowed to be directly used by application without copying data between HPNL buffer and application buffer.
  - Thanks to RDMA, it supporting user-space to kernel-space zero-copy.
- **Threading model**
  - Implements the Proactor model.
  - Interrupt + polling approach to optimize HPNL thread.
  - Supports thread binding to specific core.
- **HPNL interface**
  - C/C++ and Java interface.
  - Supports send, receive, remote read, remote write semantics.
  - Pluggable buffer management interface.
  - Capable of using Persistent memory as RDMA buffer.
- **Open Source**
  - HPNL is Under internal opensource process, expected to be opensourced in Q2.



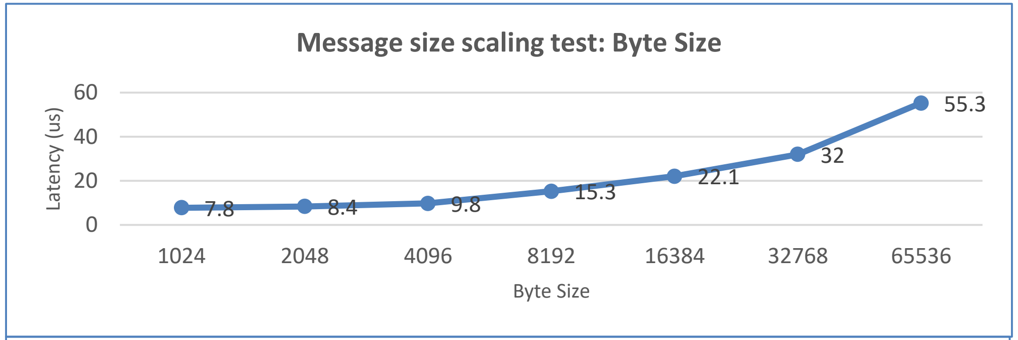
MICROBENCHMARK CONFIGURATION

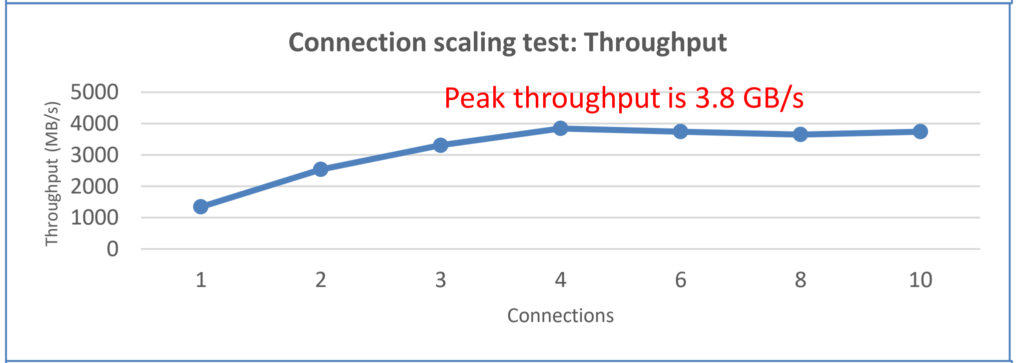
- **Hardware**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  - Mellanox ConnectX-4 – RoCE V2
  - Arista 7060 CX2-32S
- **Software**
  - HPNL Java interface with libfabric v1.6.0
  - OFED
  - CentOS 7
- **Micro benchmark Methodology**
  - Send/receive based Ping-pong test
  - 1million times transfer of 4K sized message

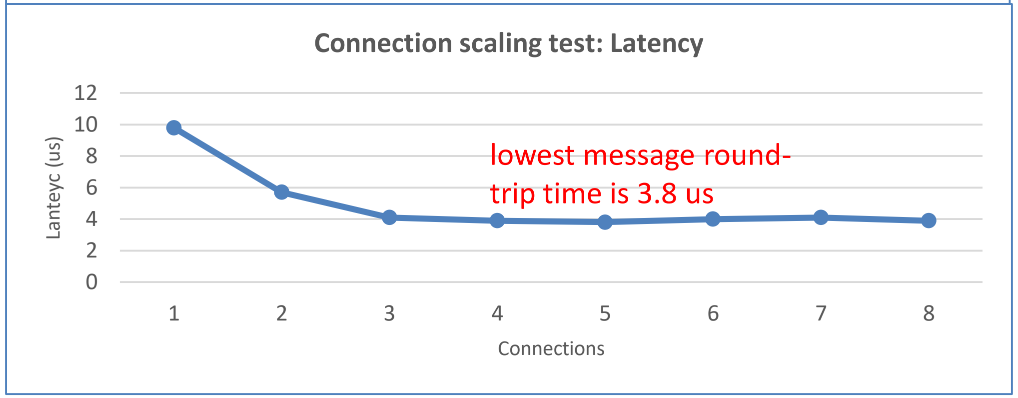


MICRO WORKLOADS PERFORMANCE

- **Test 1: message size scaling test, single thread**
  - Shows better performance than Netty or ASIO.
- **Test 2: Connection scaling test**
  - Message size: 65536
    - Hits bandwidth limitation.
  - Message size: 4096
    - Send/recv based interface shows round-trip time less than 4 us.

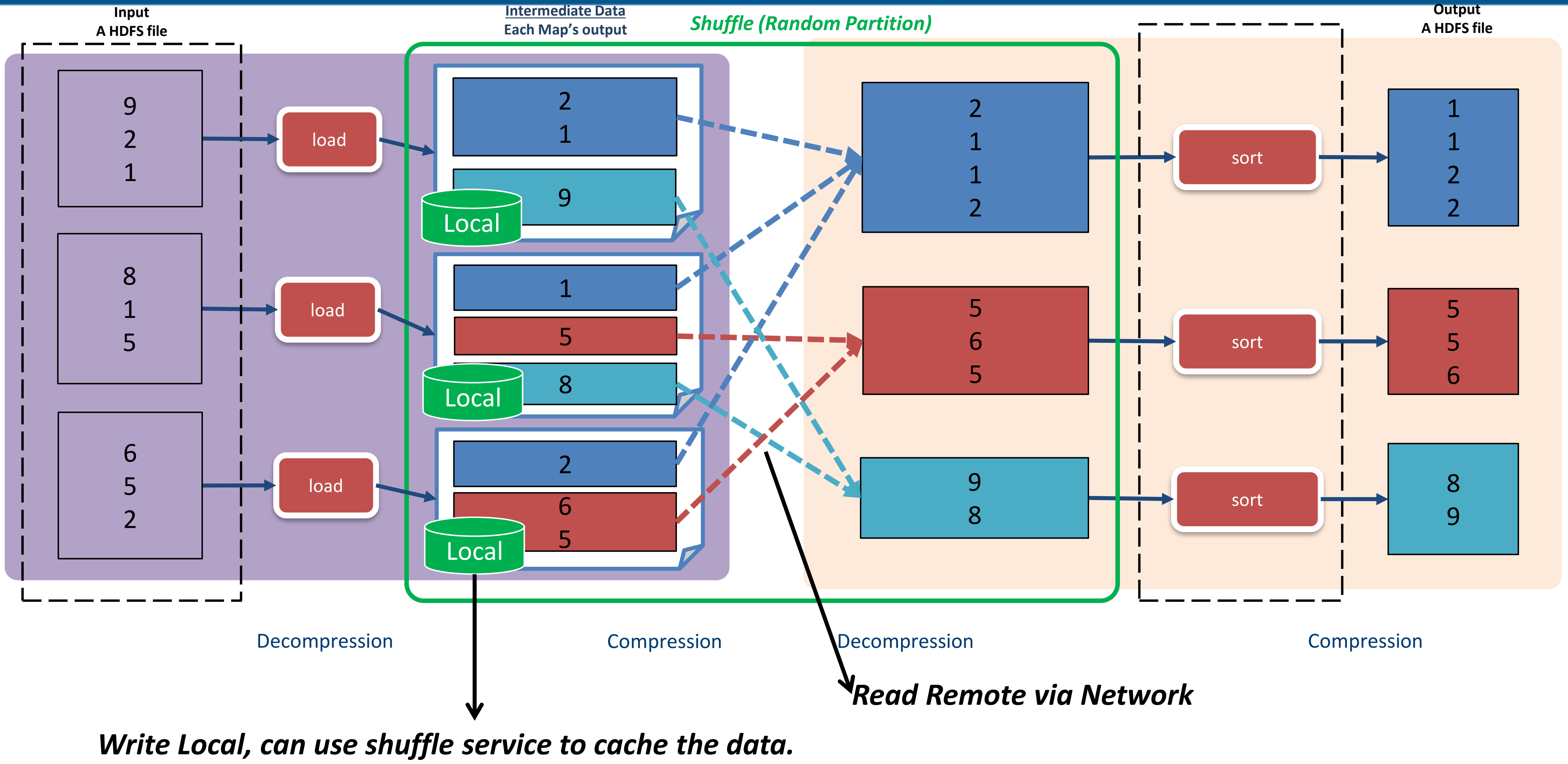






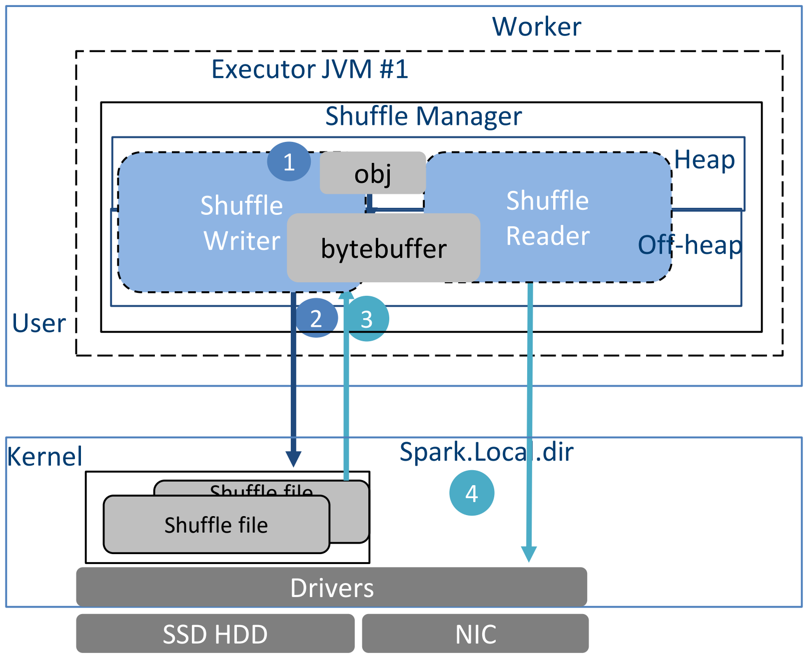
OPTIMIZING SPARK SHUFFLE WITH HPNL AND PERSISTENT MEMORY

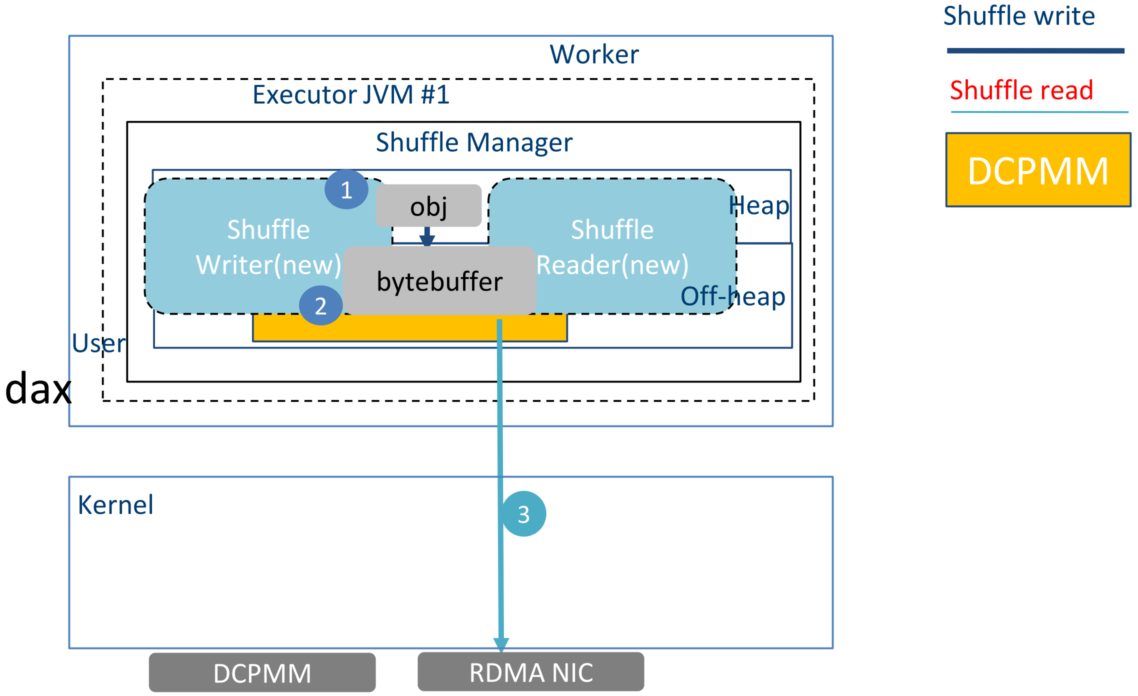
SPARK SHUFFLE



https://github.com/intel-hadoop/HiBench/blob/master/sparkbench/micro/src/main/scala/com/intel/sparkbench/micro/ScalaSort.scala

SPARK-PMOF CO-DESIGN





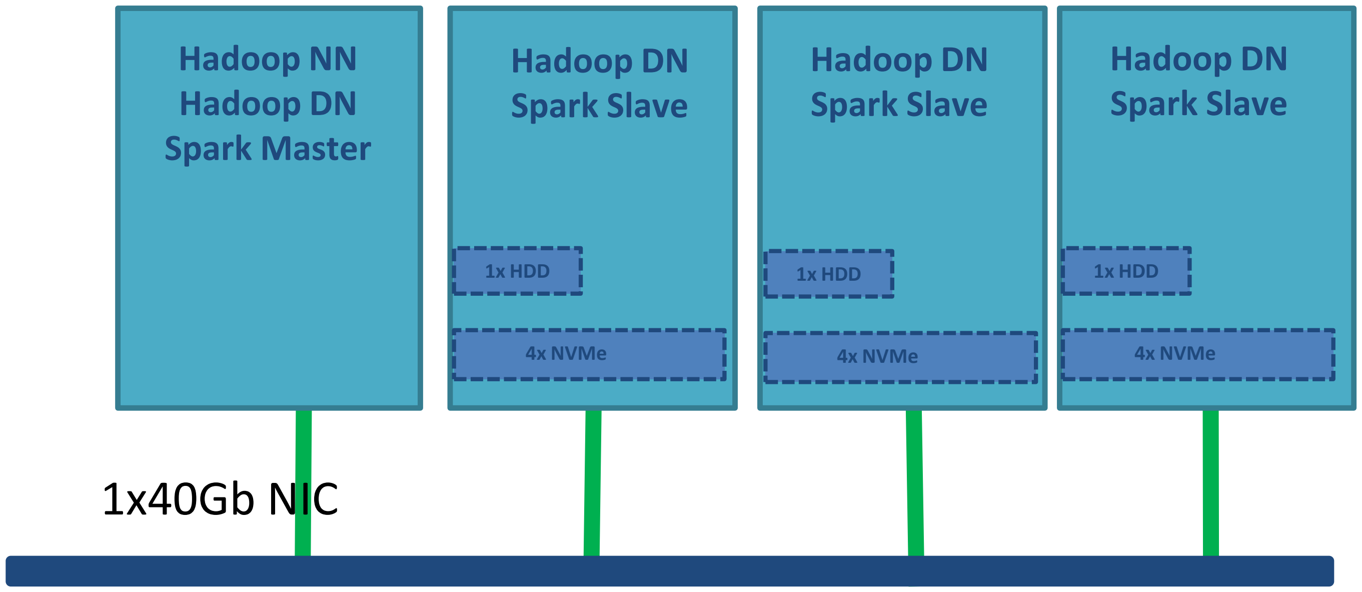
1. Serialize obj to off-heap memory
2. Write to local shuffle dir
3. Read from local shuffle dir
4. Send to remote reader through TCP-IP

- ➢ Lots of context switch
- ➢ POSIX buffered read/write on shuffle disk
- ➢ TCP/IP based socket send for remote shuffle read

1. Serialize obj to off-heap memory
2. Persistent to DCPMM (pmem_memcpy_persistent)
3. Read from remote DCPMM through RDMA, DCPMM is used as RDMA memory buffer

- ➢ No context switch
- ➢ Efficient read/write on DCPMM
- ➢ RDMA read for remote shuffle read

SYSTEM CONFIGURATION



3 Node cluster

Hardware:

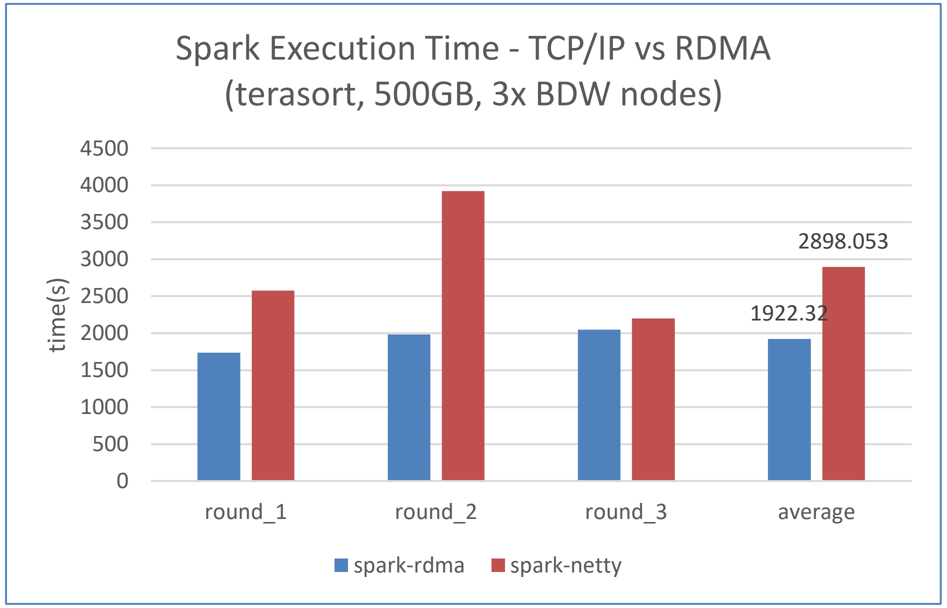
- Intel® Xeon™ processor E5 2699V4 @ 2.2GHz, 558GB Memory
- 1x Mellanox ConnectX-4 40Gb NIC
- 1x 1TB HDD for spark-shuffle
- 4x NVMe for HDFS

Software:

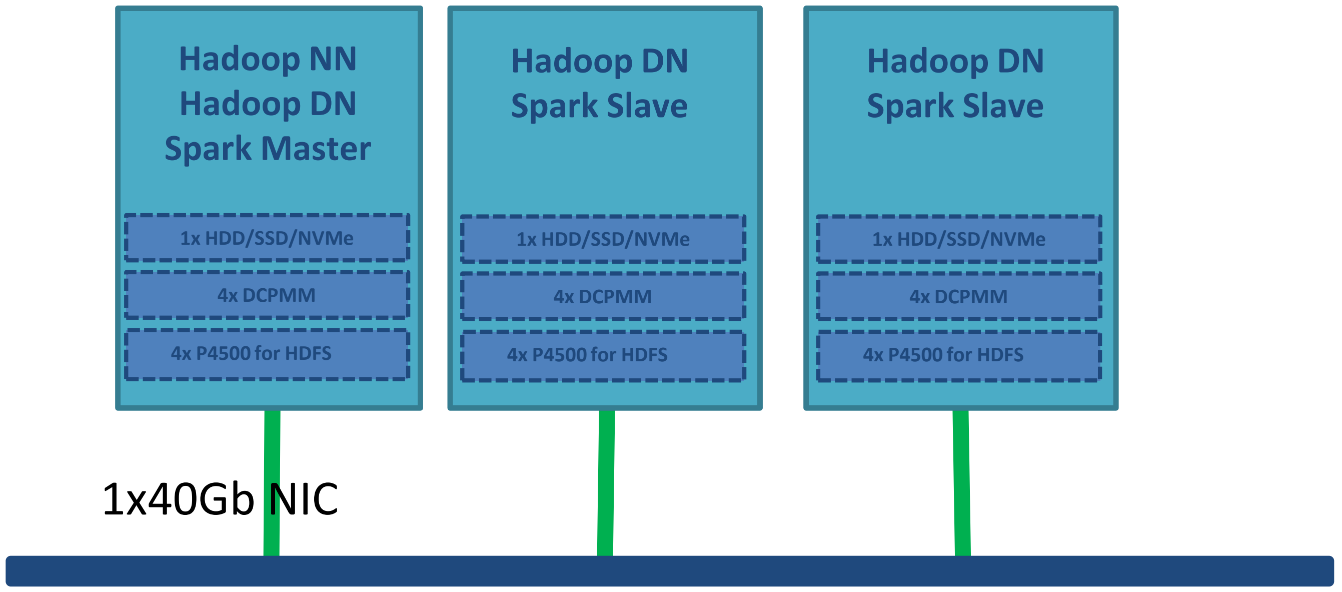
- Hadoop 2.7
- Spark 2.3
- CentOS 7
- HiBench TeraSort 500GB

RDMA PERFORMANCE BENEFITS FOR SPARK SHUFFLE

- **1.5x performance improvement**
  - And much stable!
  - Page cache impact on spark-netty performance



SYSTEM CONFIGURATION



3 Node cluster

Hardware:

- Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz
- 1x Mellanox ConnectX-4 40Gb for shuffle
- 384 G MEM
- 1 x X722 for Hadoop
- 4x P4500 for HDFS
- 4x 256GB DCPMM
- 1x HDD/SSD (1x 400Gb DC S3700)/NVMe (1x P4500)/(4x 256GB) DCPMM for Shuffle

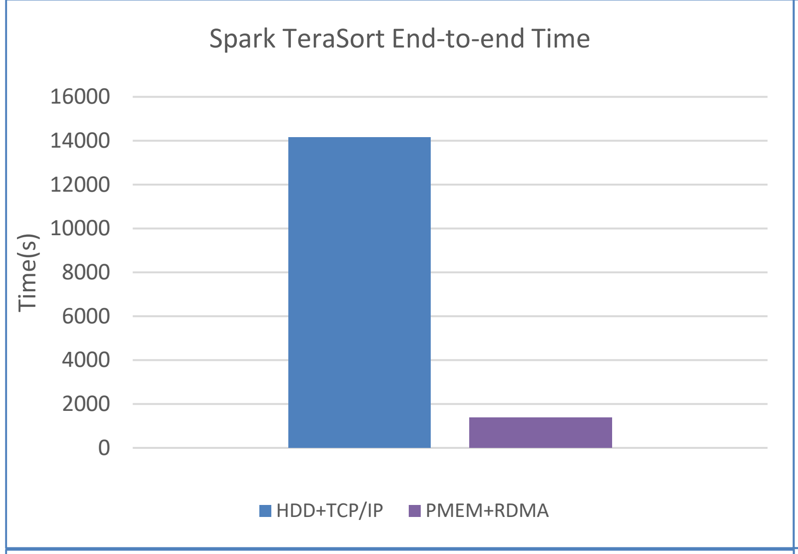
Software:

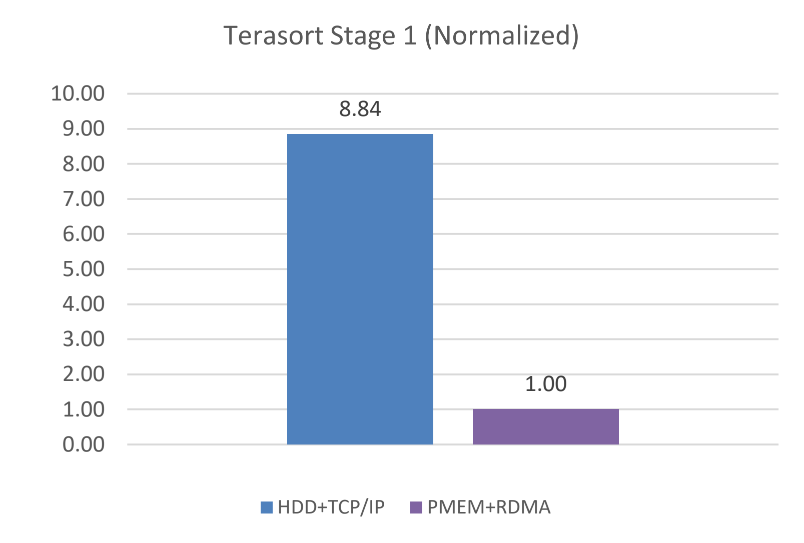
- Fedora 27
- Kernal 4.18.19-100.fc27.x86_64
- Hadoop 2.7
- Spark 2.3
- Libfabric 1.6.2
- MLNX_OFED_LINUX-4.5-1.0.1.0-fc27-x86_64

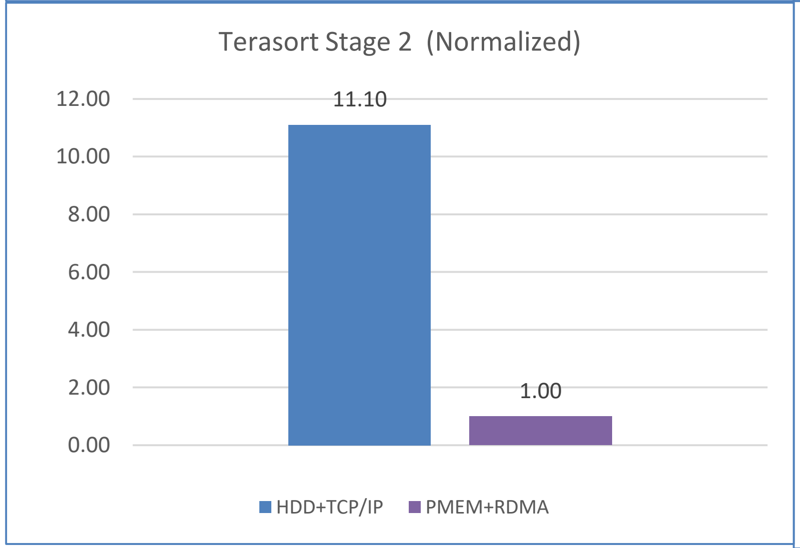
Workloads:

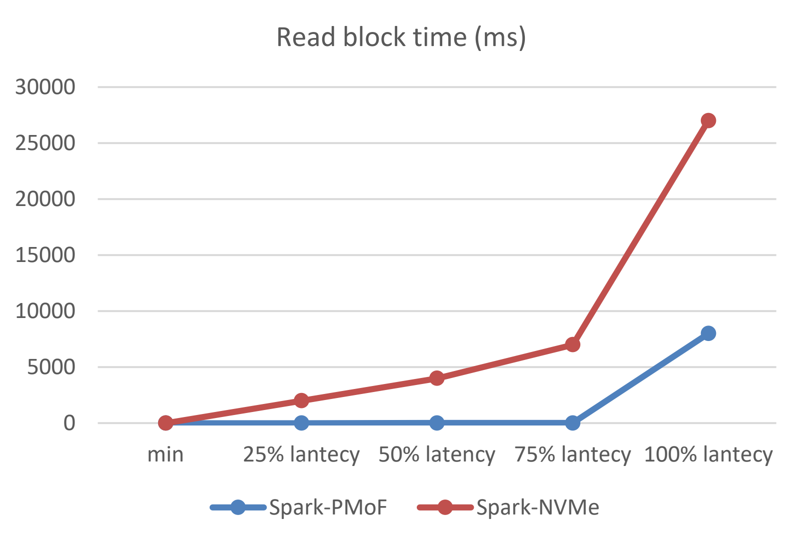
- Hibench terasort 1TB

PERFORMANCE SUMMARY









- **PMoF vs. HDD**
  - PMEM and PMEM+RDMA are 9.17x and 9.10x faster than HDD respectively.
  - However, the benefit over NVMe+TCP/IP is not very big
  - Other opportunities:
    - Lower CPU utilization
    - Latency sensitive workloads – streaming

STAGE 2 PERFORMANCE BREAKDOWN (HDD+TCP/IP)

Summary Metrics for 1000 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3.7 min	11 min	13 min	14 min	22 min
Scheduler Delay	0 ms	0 ms	2 ms	3 ms	23 ms
Task Deserialization Time	2 ms	3 ms	3 ms	4 ms	1.0 s
GC Time	4 s	8 s	9 s	11 s	21 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Output Size / Records	632.9 MB / 6636259	874.1 MB / 9166014	951.7 MB / 9979334	1035.6 MB / 10858719	1329.5 MB / 13940317
Shuffle Read Blocked Time	2.5 min	8.5 min	9.7 min	11 min	15 min
Shuffle Read Size / Records	670.9 MB / 6636259	926.6 MB / 9166014	1008.8 MB / 9979334	1097.7 MB / 10858719	1409.2 MB / 13940317
Shuffle Remote Reads	596.7 MB	823.5 MB	896.0 MB	976.8 MB	1253.5 MB

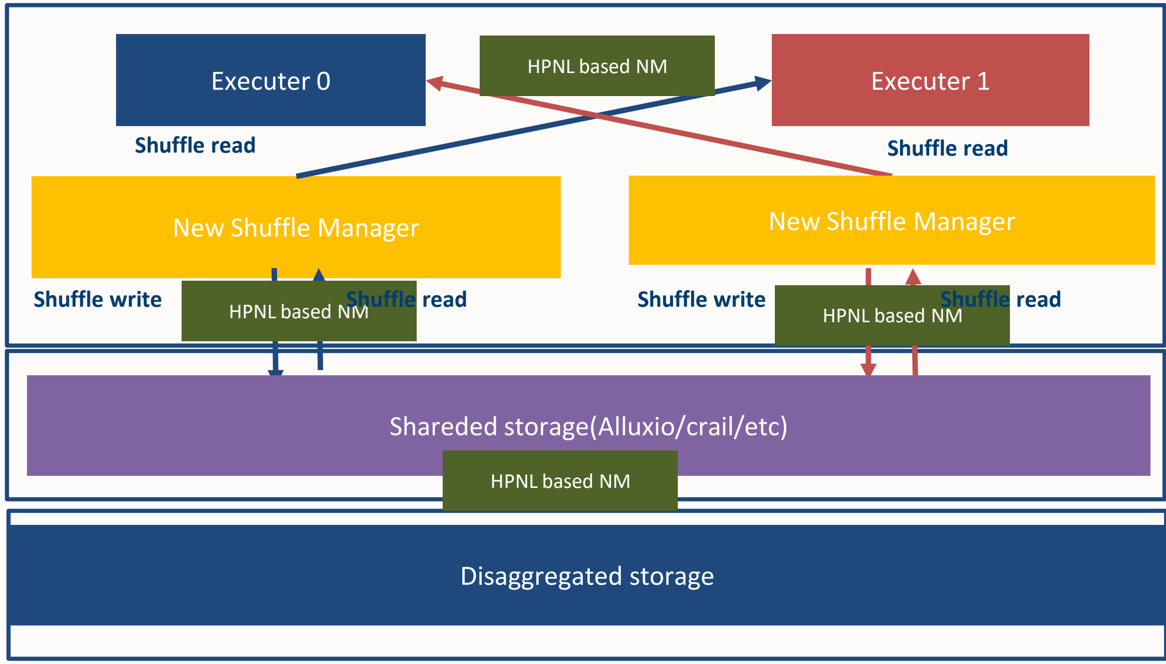
STAGE 2 PERFORMANCE BREAKDOWN (PMEM+RDMA)

Summary Metrics for 1000 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	36 s	1.0 min	1.1 min	1.3 min	1.7 min
Scheduler Delay	0 ms	3 ms	3 ms	4 ms	0.2 s
Task Deserialization Time	2 ms	4 ms	5 ms	6 ms	0.1 s
GC Time	2 s	9 s	11 s	13 s	26 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	38 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Output Size / Records	632.9 MB / 6636259	874.1 MB / 9166014	951.7 MB / 9979334	1035.6 MB / 10858719	1329.5 MB / 13940317
Shuffle Read Blocked Time	0 ms	2 ms	3 ms	5 ms	8 s
Shuffle Read Size / Records	670.9 MB / 6636259	926.6 MB / 9166014	1008.8 MB / 9979334	1097.7 MB / 10858719	1409.2 MB / 13940317
Shuffle Remote Reads	598.3 MB	820.3 MB	895.6 MB	976.8 MB	1256.8 MB

NEXT: HPNL INTEGRATION WITH EXTERNAL SHUFFLE CLUSTER

- **Elastic Deployment with compute and storage disaggregation requires independent shuffle solution**
  - Shuffle I/O are decoupled from a specific network/storage.
  - Shuffle read and write can be implemented using configurable network transports and backend storage
- **External shuffle cluster**
  - Lots of on-going efforts making HCFS as shuffle: [Spark-1529, Spark-3685, SPARK-25299]
  - And lots of customized solutions
  - Independent HPNL library can be integrated with customized solutions
  - Opensource version reference design based on HCFS shuffle manager and HDFS will come out soon



SUMMARY

SUMMARY & NEXT STEP

- **Summary**
  - Traditional TCP/IP stack can't benefit more and more Real-time BigData processing even with modern network hardware while RDMA has been proved as an effective way to speedup BigData workload from existing projects.
  - HPNL is a transportation Agnostic high performance network library based on libfabric
  - HPNL demonstrated significant performance advantage over TCP/IP for bigdata applications
    - 1.5x over TCP/IP for spark terasort
  - Spark shuffle with HPNL and pmem delivers up to 9x performance improvement over traditional shuffle solution based on TCP/IP and HDD
  - Persistent memory over fabrics with HPNL and pmem enables new workloads and new storage solutions
    - Latency sensitive workloads and external/remote shuffle cluster
- **Next step**
  - RPC with HPNL
  - RDMA in external shuffle cluster

15th ANNUAL WORKSHOP 2019

THANK YOU

Haodong Tang, Jian Zhang, Fred Zhang

haodong.tang@intel.com, jian.zhang@intel.com, fred.zhang@intel.com

Intel

[March, 2019]

BACKUP

BACKUP – SPARK PMOF TEST CONFIGURATIONS

- **Terasort 1TB:**
- hibench.spark.master   yarn-client
- hibench.yarn.executor.num   12
- yarn.executor.num   12
- hibench.yarn.executor.cores   8
- yarn.executor.cores   8
- spark.shuffle.compress   false
- spark.shuffle.spill.compress   false
- spark.executor.memory 60g
- spark.executor.memoryoverhead 10G
- spark.driver.memory   80g
- spark.eventLog.compress = false
- spark.executor.extraJavaOptions=-XX:+UseG1GC
- spark.hadoop.yarn.timeline-service.enabled false
- spark.serializer   org.apache.spark.serializer.KryoSerializer
- hibench.default.map.parallelism   200
- hibench.default.shuffle.parallelism 1000