SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

RPMP: A Remote Persistent Memory Pool to accelerate data analytics and AI

Jian Zhang
Intel

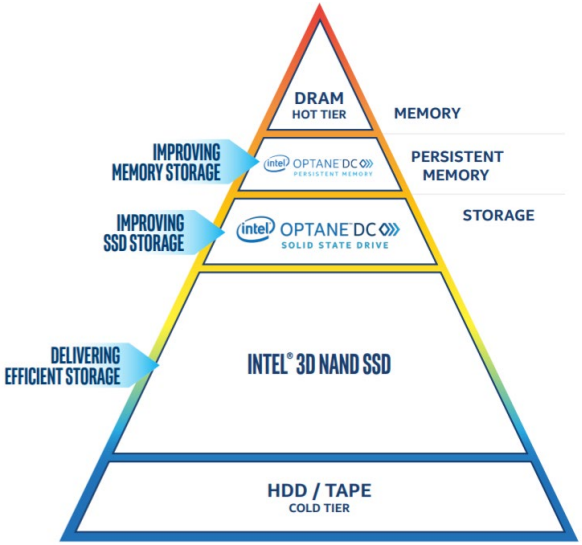
Agenda

- Persistent Memory Introduction
- Remote Persistent Memory & Usage
- Remote Persistent Memory Pool Design and implementation
- Remote Persistent Memory Pool Performance
- Summary

Persistent Memory

Persistent Memory - A New Memory Tier

- IDC reports indicated that data is growing very fast
  - Global datasphere growth rate (CAGR) 27%*
  - But DRAM density scaling is becoming slower: from 4X/3yr (1997-ish) to 2X/3yr (~1997-2010) to 2X/4yr** (since 2010)
  - A new memory system will be needed to met the data growth needs for new cases
- Persistent Memory (PMem): new category that sits between memory and storage
  - Delivers a unique combination of affordable large capacity and support for data persistence



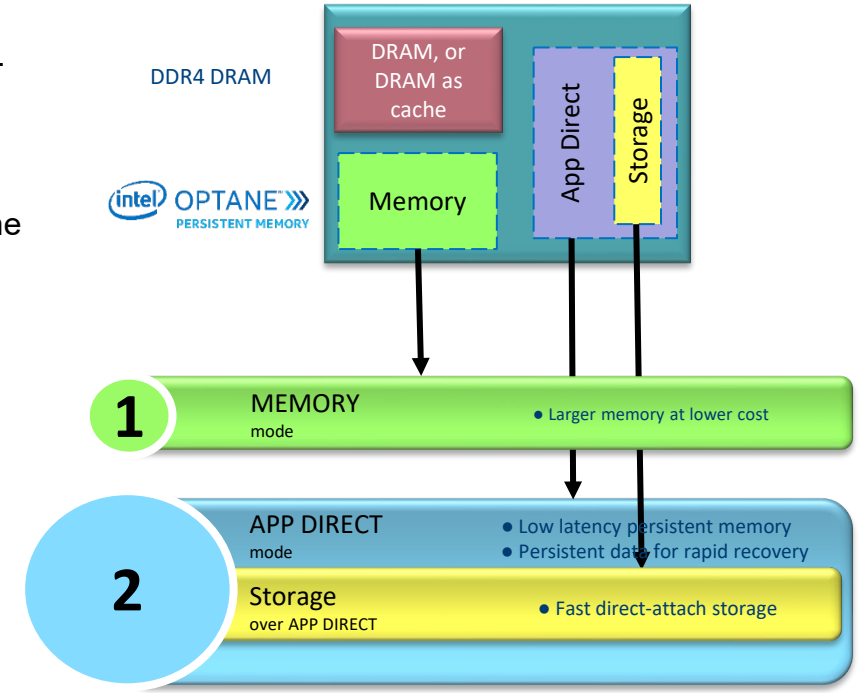
Storage Hierarchy***

*Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018
**Source: "3D NAND Technology – Implications for Enterprise Storage Applications" by J.Yoon (IBM), 2015 Flash Memory Summit
*** https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dc-persistent-memory-brief.html

Persistent Memory

Operation Mode

- Memory Mode
  - Volatile storage. Provides higher memory capacity. Cache management is handled by processor's integrated controller.
- APP Direct Mode
  - Non-volatile storage. Application manages its cache data by itself. Read/write bypass page cache.
- Characteristics:
  - DDR4 electrical & physical
  - Close to DRAM latency
  - Cache line size access
  - 128/256/512GB for single PMem

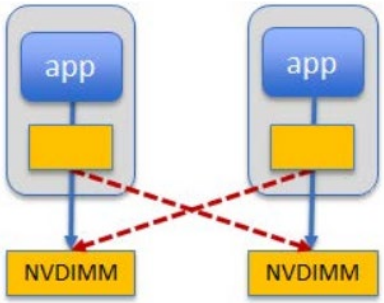


* https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

Remote Persistent Memory

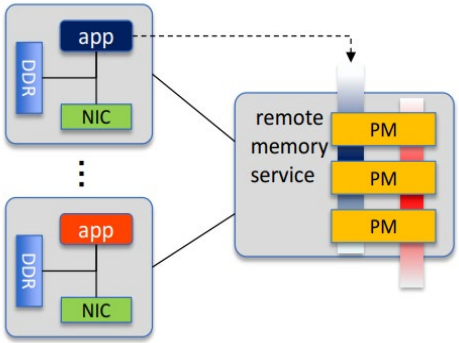
Remote Persistent Memory Usage

High Availability Data Replication



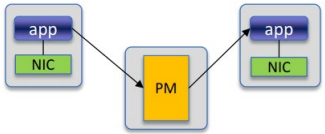
- Replicate Data in local PM across Fabric and Store in remote PM
- Improved redundancy

Remote PM



- Extend on-node memory capacity (w/ or w/o persistency) in a disaggregated architecture to enlarge compute node memory
- Increased capacity

Shared Remote PM



- A PM pool holds sharded data for distributed applications
- Improved performance

https://www.snia.org/sites/default/files/PM-Summit/2018/presentations/05_PM_Summit_Grun_PM_%20Final_Post_CORRECTED.pdf

Remote Persistent Memory over RDMA

Remote Persistent Memory offers

- Remote persistence, without losing any of characteristic of memory
- PM is Fast
  - Needs ultra low-latency networking
- PM has very high bandwidth
  - Needs ultra efficient protocol, transport offload, high BW
- Remote access must not add significant latency
  - Network switches & adaptors deliver predictability, fairness, zero packet loss

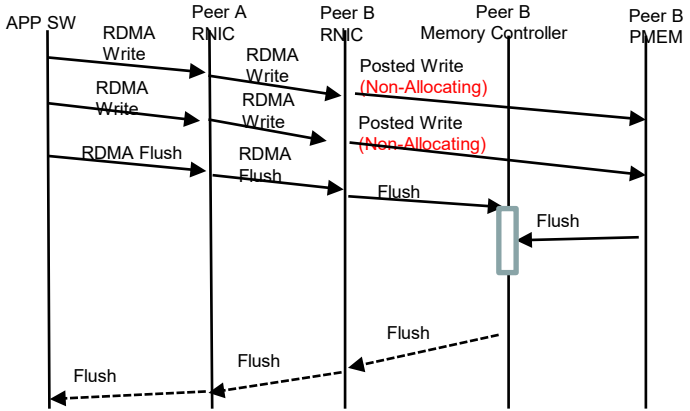
RDMA offers

- Moving data between (zero-copy) two system with Volatile DRAM, offload data movement from CPU to NIC
- Low latency
  - Latency < uses
- High BW
  - 200Gb/s, 400Gb/s, zero-copy, kernel bypass, HW offered one side memory to remote memory operations
- Reliable credit base data and control delivered by HW
  - Network resiliency, scale-out

RPMem Durability

- RPMem over Fabric additional complexity:
  - In order to guarantee written data is durable on the target node, CPU caches need to be bypassed or flushed to get data into the ADR power fail safe domain
  - When writing to PMEM, need a synchronous acknowledgement when writes have made it to the durability domain but lacks in current RDMA Write semantics
- RDMA
  - Guarantee that Data has been successfully received and accepted for execution by the remote HCA
    - Doesn't guarantee data has reached remote host memory – need ADR
    - Doesn't guarantee the data can be visible/durable for other consumers accesses (other connections, host processor)
    - Using small RDMA read to forces write data to PMem

- New transport operation – **RDMA FLUSH**
  - Flush all previous writes or specific regions
  - Provides memory placement guarantee to the upper layer software
  - RDMA Flush forces previous RDMA Write data to durability domain
- It makes PM operations with RDMA more efficient!

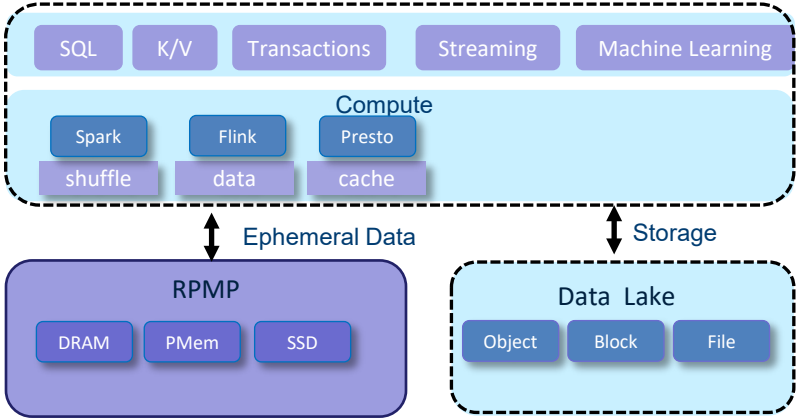


RPMP

RPMP motivations

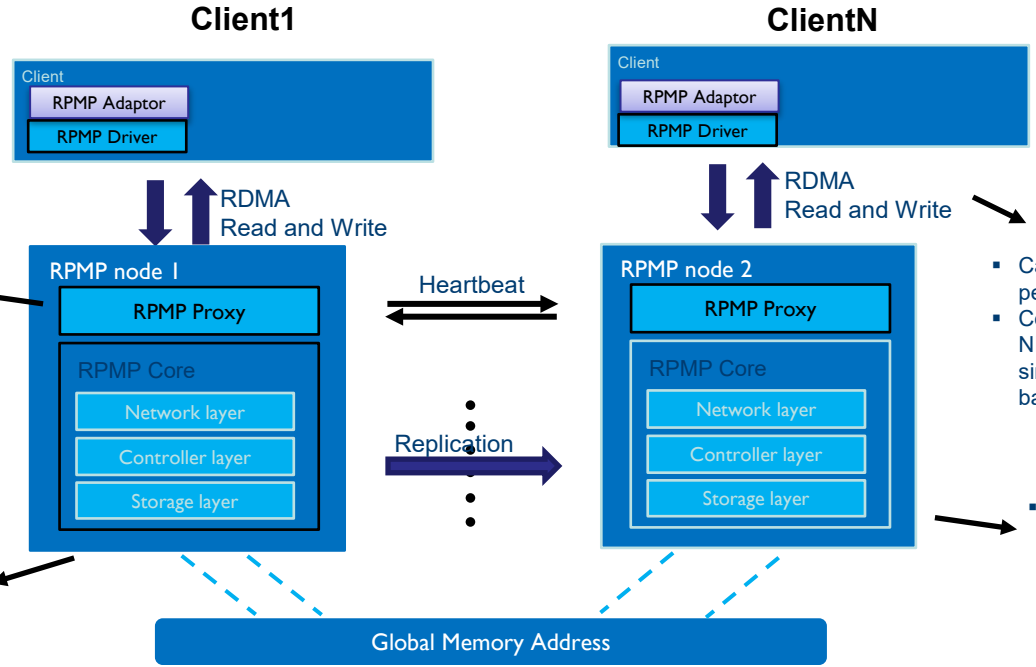
- Challenges in real production analytics and AI cluster
  - Memory is a precious resource, always insufficient
  - CPU/Memory/Storage unbalanced issue
  - Adopt compute storage disaggregation for better scalability and cost saving
- Scale compute & storage independently typically requires:
  - High performance, low latency distributed storage w/ rich APIs
  - Centralized high-performance distributed cache shared by various applications
  - Fault tolerant to avoid stage recompute for long running analytics jobs
- Remote Persistent Memory pool
  - A distributed storage solution based on PMem
  - Target for high performance storage or accelerate ephemeral data access
  - Leveraging RDMA for remote PMem access
  - Rich APIs for different usage scenarios

RPMP Architecture



- Remote Persistent Memory Pool:
  - A persistent memory based distributed storage system
  - An RDMA powered network library and an innovative approach to use persistent memory as both shuffle media as well as RDMA memory region to reduce additional memory copies and context switches.
  - Target as high-performance storage & ephemeral data storage
- Features
  - High Performance Storage powered by modern HW like PMem and RDMA w/ user-level I/O access
  - Rich API: Provides memory-like allocate/free/read/write APIs on pooled PMem resources
  - Pluggable modules: A modular architecture makes it can be plugged into in memory database, Apache Spark (shuffle) & Cache etc.
  - Heterogeneous tiered storage backend
- Benefits
  - Improved scalability of analytics and AI workloads by disaggregating ephemeral data from compute node to a high-performance distributed storage, e.g., Spark shuffle
  - Improved performance with high speed persistent memory and low latency RDMA network
  - Improved reliability by providing a manageable and highly available disaggregated storage supports ephemeral data replication and fault-tolerant, e.g. shuffle data to avoid recompute.

Remote Persistent Memory Pool overview



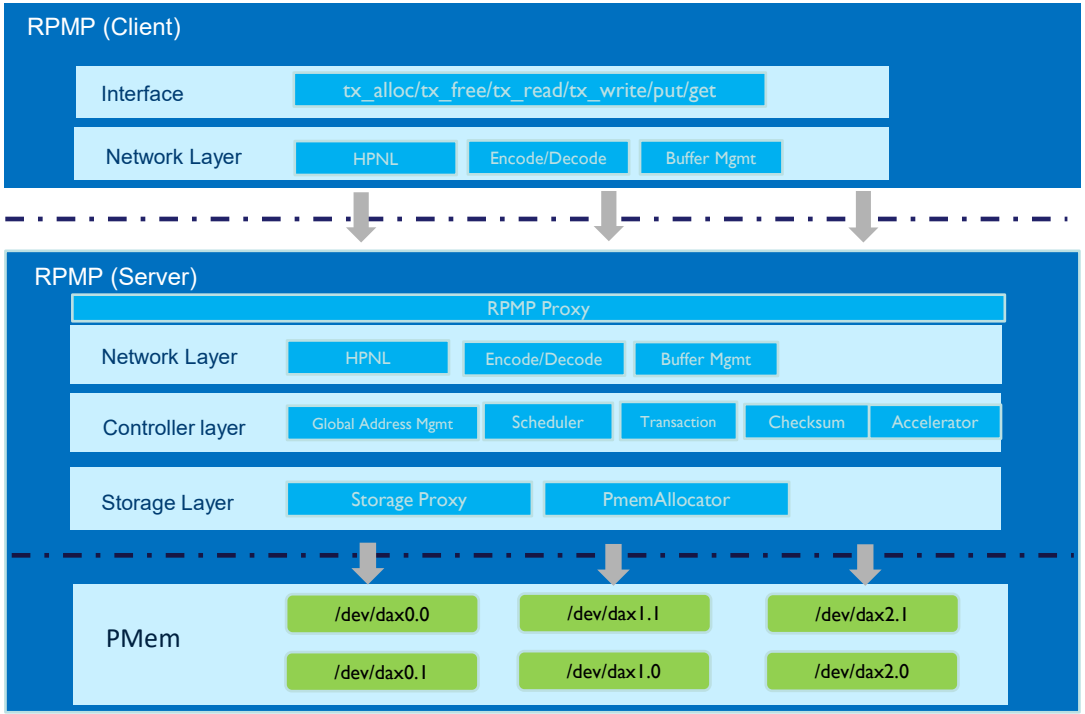
- RPMP storage node was choosen by using **configured distribution policy** to avoid single point failure.
- A timely **ActiveNodeMap** maintained by using Heartbeat.

- Data will be replicated to a Secondary node from primary node over RDMA

- Care more about write performance (latency/bandwidth)
- Combined with high performance NIC, theoretically 8x PMEM on single node provide 10GB+ write bandwidth.

- If primary node goes down, secondary node is still writable and readable.

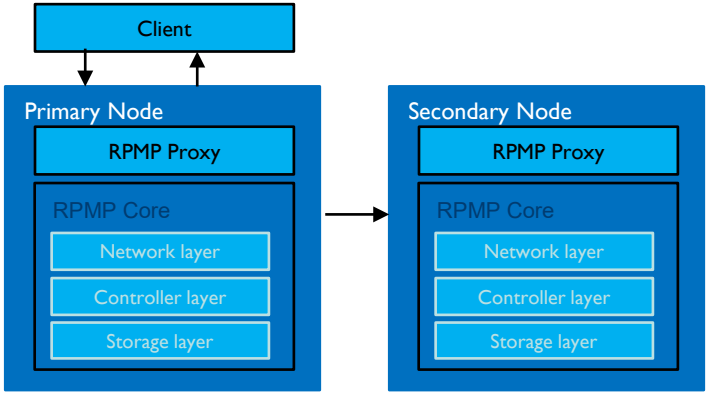
RPMP architecture details



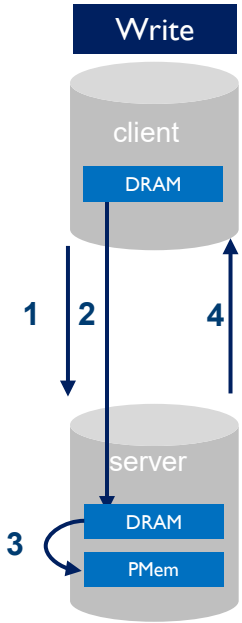
- RPMP Client
  - RPMP client provides transactional read/write/allocate/free, and obj put/get interfaces to users
  - Both cpp and java API are provided
  - Data will be transferred by HPNL(RDMA) between selected server nodes and client nodes.
- RPMP Server
  - RPMP proxy is used to maintain an unified ActiveNodeMap and distributes client request among RPMP nodes.
  - Network Layer is based on HPNL to provide RDMA Data transfer.
  - Controller Layer is responsible for Global Address Management, TransactionalProcess, etc.
  - Storage Layer is responsible for Pmem management using high performance user space PMDK libs

RPMP Proxy

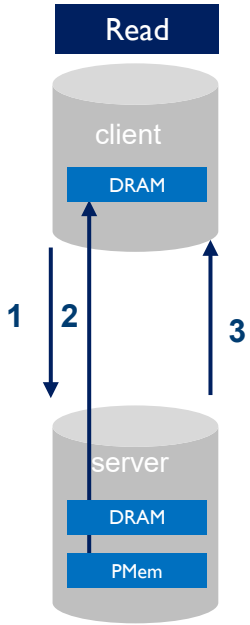
- Cluster-level node map
  - Heartbeat between RPMP node and proxy nodes to maintain activenodemap.
  - Heartbeat between Primary node and secondary node to check health status of each RPMP node
- Configurable distribution policy
  - Static configuration file for Primary/secondary pair
  - Consistency Hash algorism w/ active nodemap
- Primary/Secondary architecture for Data Replication
  - Client contacts proxy for primary node
  - Client sends write to Primary, then Primary replicates to secondary, Client ACK on both Primary write and replication write finished.
- Lazy failover & recovery
  - Do not attempt to launch failed node or recover data, let admin make the decision



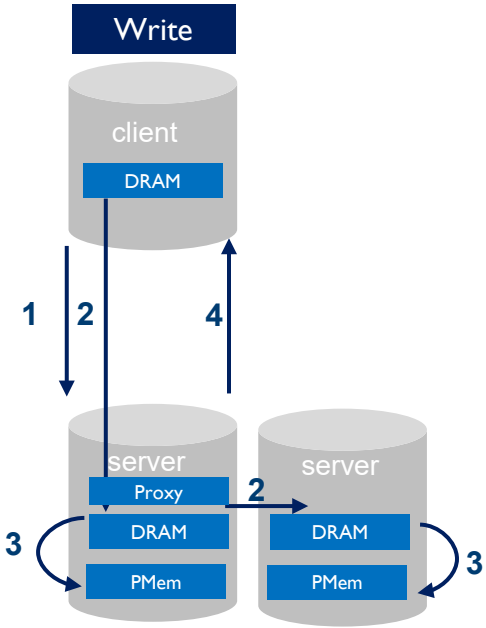
RPMP Read Write Flow



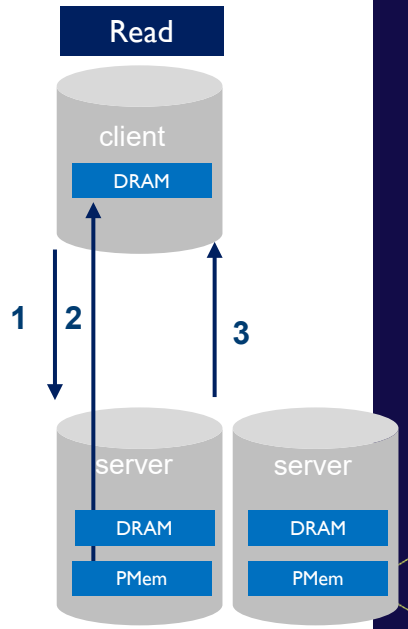
1. Write data to specific address.
2. Server issue RDMA read (client DRAM -> server DRAM).
3. Flush (DRAM -> PMEM).
4. Request ACK.



1. Read data from specific address.
2. RDMA write (server PMEM -> client DRAM).
3. Request ACK.



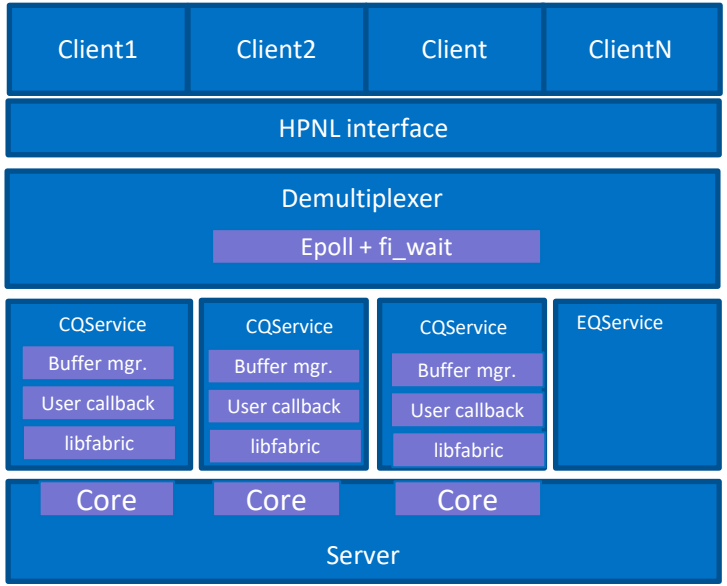
1. Write data to specific address.
2. RDMA read (client DRAM -> server DRAM), Secondary Node DRAM -> Primary Node DRAM)
3. Flush (DRAM -> PMEM).
4. Request ACK.



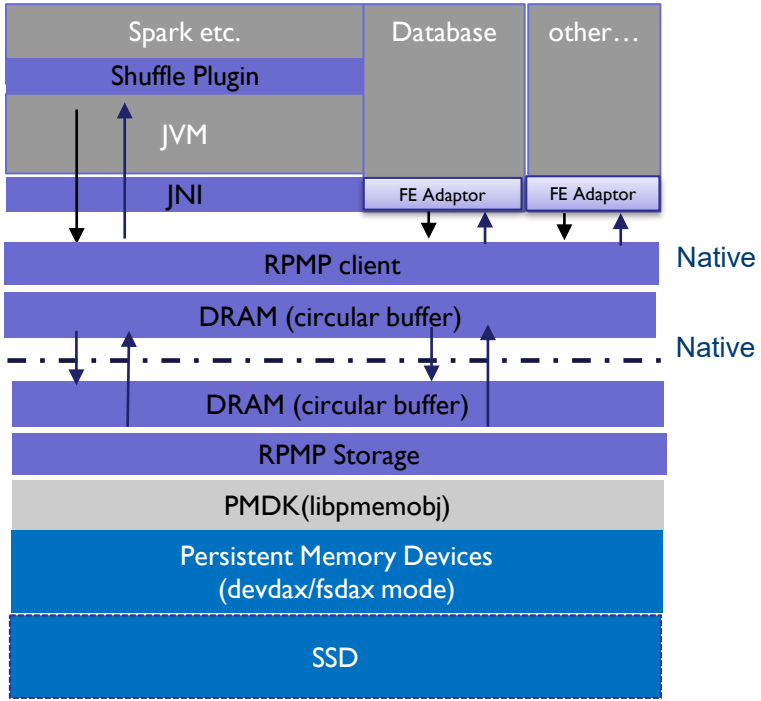
1. Read data from specific address.
2. RDMA write (server PMEM -> client DRAM).
3. Request ACK.

RPMP - Network

- Zero-copy approach
  - The HPNL buffer allowed to be directly used by application without copying data between HPNL buffer and application buffer.
  - 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 Sourced
  - https://github.com/intel-bigdata/hpnl

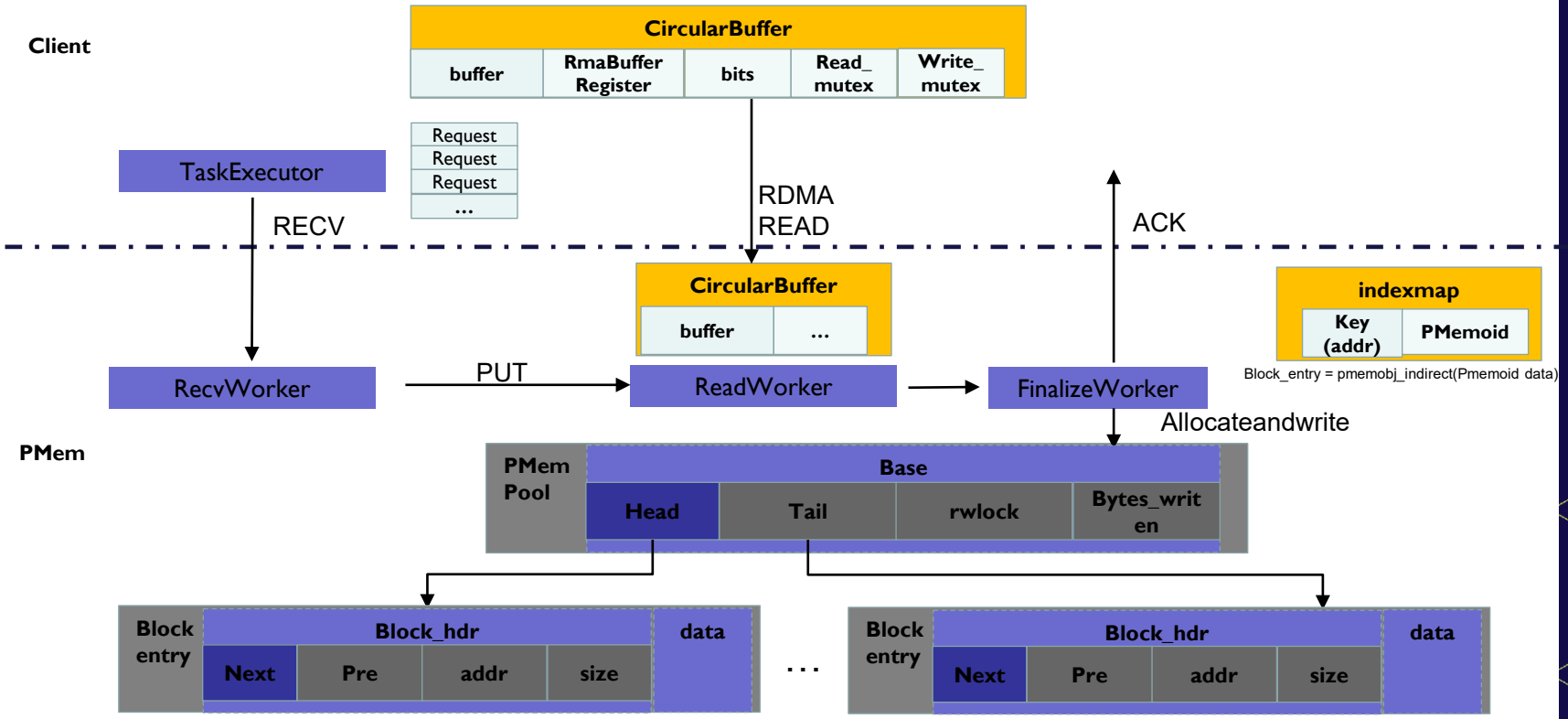


RPMP Storage



- RPMP Storage
  - A transactional and atomic Key Value Data Store w/ user-space access, designed to fully drive PMem capabilities
  - Extend: backend adaptor to tier data to different storage media, e.g. DRAM, PMem, SSD, HDD
- Space Management
  - Provision pmem name space in advance
  - Leverage a circular buffer to build unidirectional channels for RDMA primitives
  - **For Write**
    - Libpmemobj converts PMem to flexible object store
    - Data write to a circular buffer, once hit threshold (4MB by default), create a block via libpmemobj on pmem device with memcopy
    - Append write, only write once
    - Tiering support planned
  - **For read :**
    - Use memcopy to read the data
    - Read through PMem memory directly from RDMA memory region that registered on PMem (avoid DRAM to PMem copies)
  - No index file, mapping info stored in pmem object metadata
  - Libpmem based, kernel bypass

RPMP Data layout

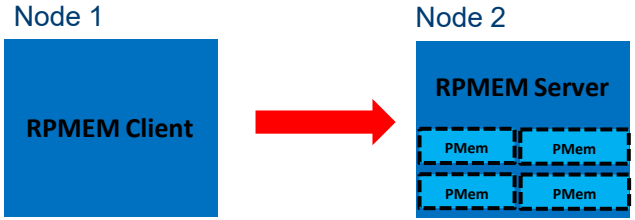


Other RPMP features

- Optimized RDMA communication
  - Leverage HPNL as high performance, protocol agnostic network messenger
  - Server handle all the write operations and clients implement read-only operations using one-sided RDMA reads.
- Controller accelerator layer
  - Partition Merge
    - Aggregate small partitions to larger blocks to accelerate reduce, reduced number of reducer connections
  - Sort
    - Sort the shuffle data on the fly, no compute on reduce phase, reduce compute node CPU resource utilization
    - Provided controllable fine granularity control or resource utilization if compute node CPU resources is limited
- Storage
  - Global address space accessible with memory-like APIs
  - A Key-value store based on libpmemobj, transactional
  - Allocator to manager on PMem, storage proxy directing request to different allocators

RPMP Performance

RPMP KV Storage Performance



Interface	Performance	
allocate	3.7 GB/s	Expect higher performance with more clients.
remote_write	2.9 GB/s	Expect higher performance with more clients.
remote_read	4.9 GB/s	Limited by 40Gb NIC

- Configuration
  - 2 nodes, one as RPMEM client and another as RPMEM server.
  - 40 GB RDMA NIC, 4x PMems on RPMEM server.
  - Tested remote_allocate, remote_free, remote_write, remote_read interfaces with single client.
- Performance
  - Remote read max out NIC BW

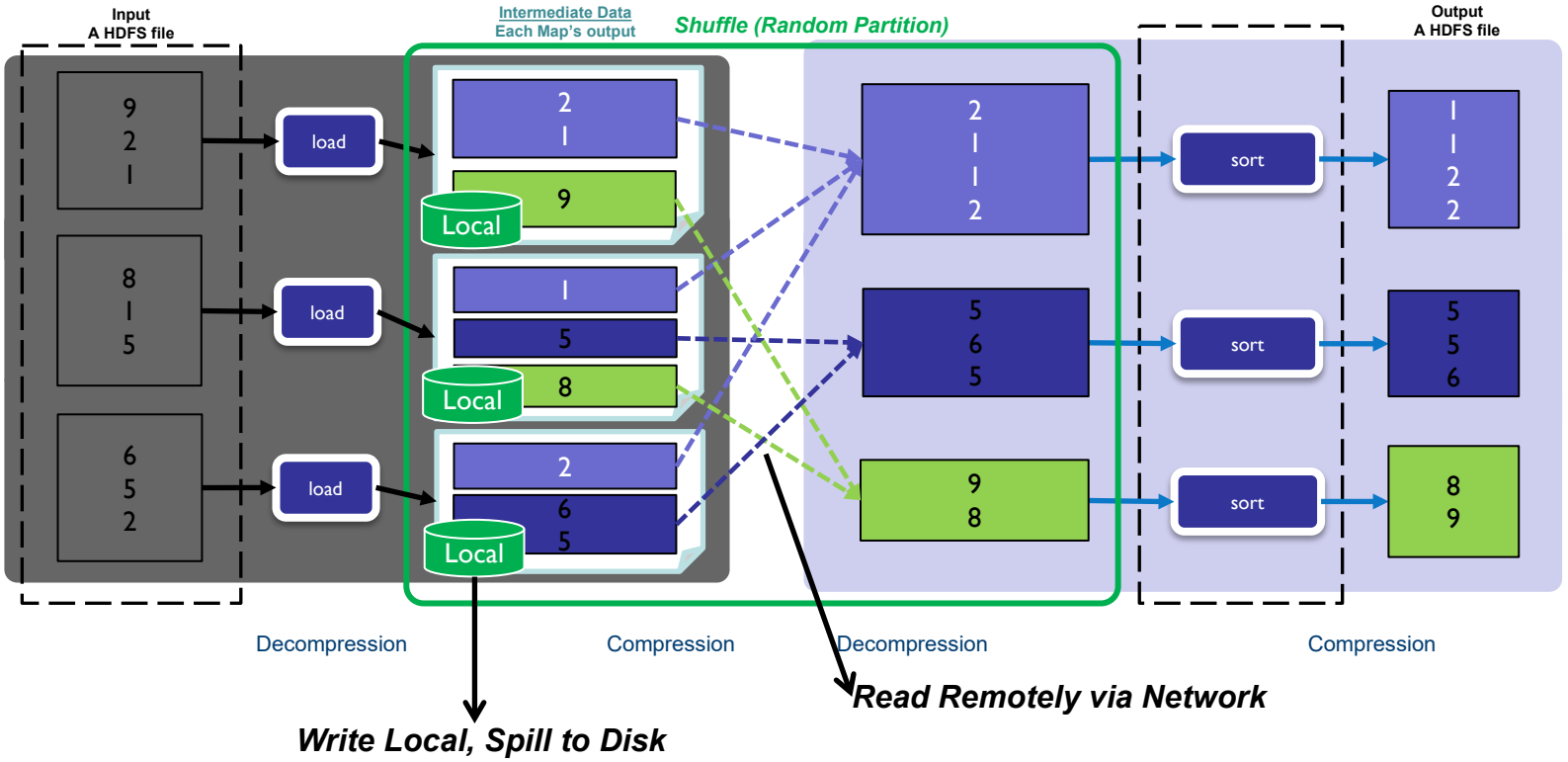
Performance results are based on testing as of 5/30/2020 and may not reflect all publicly available security updates. See configuration disclosure on slide for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. Configurations refer to page 41

RPMP for Spark shuffle – Challenges

- Data Center Infrastructure evolution
  - Compute and storage disaggregation become a key trend, diskless environment becoming more and more popular
  - Modern datacenter is evolving: high speed network between compute and disaggregated storage and tiered storage architecture makes local storage less attractive
  - New storage technologies are emerging, e.g., storage class memory (or PMem)
- Spark shuffle problems
  - Uneven resource utilization of CPU and Memory
  - Out of memory issues and GC
  - Disk I/O too slow, Data spill degrades performance
  - Shuffle I/O grows quadratically with data
  - Local SSDs wear out by frequent intermediate data writes
  - Unaffordable re-compute cost
- Other related works
  - Intel Disaggregated shuffle w/ DAOS[1], Facebook cosco[2], Baidu DCE shuffle[3], JD.com & Memverge RSS [4] and etc.

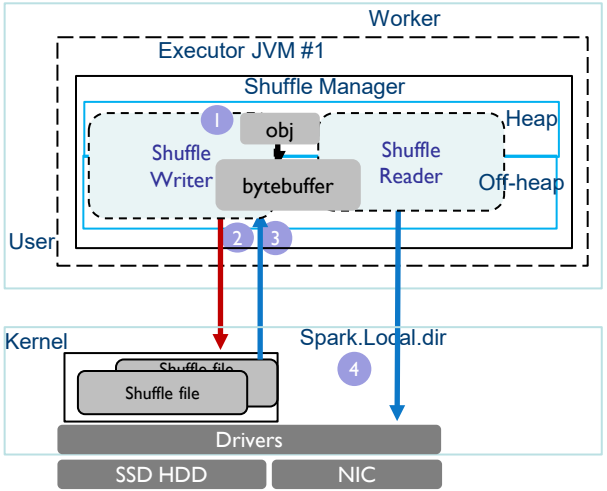
1. https://www.slideshare.net/databricks/improving-apache-spark-by-taking-advantage-of-disaggregated-architecture
2. https://databricks.com/session/cosco-an-efficient-facebook-scale-shuffle-service
3. http://apache-spark-developers-list.1001551.n3.nabble.com/Enabling-fully-disaggregated-shuffle-on-Spark-td28329.html
4. https://databricks.com/session/optimizing-performance-and-computing-resource-efficiency-of-in-memory-big-data-analytics-with-disaggregated-persistent-memory

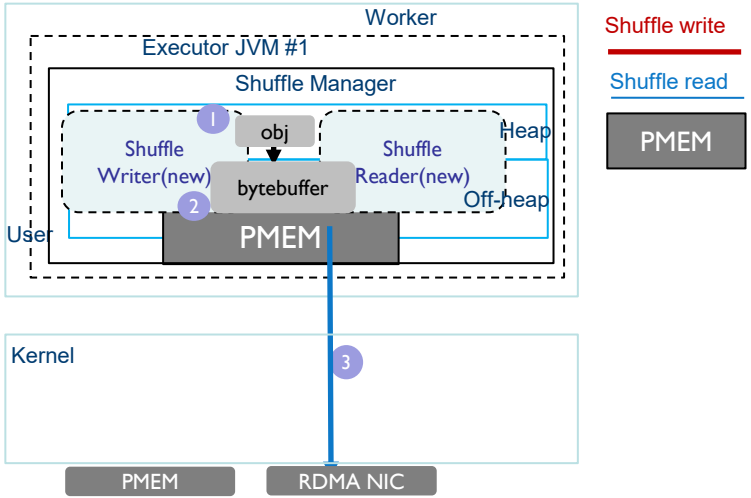
Re-cap of Shuffle



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

Re-cap: Remote Persistent Memory Extension for Spark shuffle 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

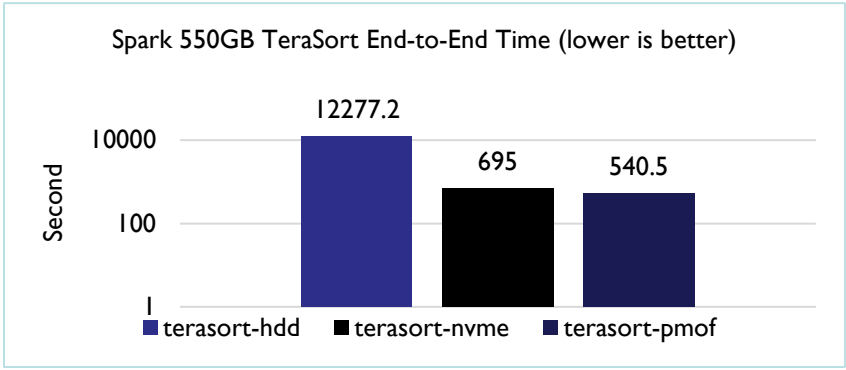
1. Serialize obj to off-heap memory
2. Persistent to PMEM
3. Read from remote PMEM through RDMA, PMEM is used as RDMA memory buffer
- No context switch
- Efficient read/write on PMEM
- RDMA read for remote shuffle read

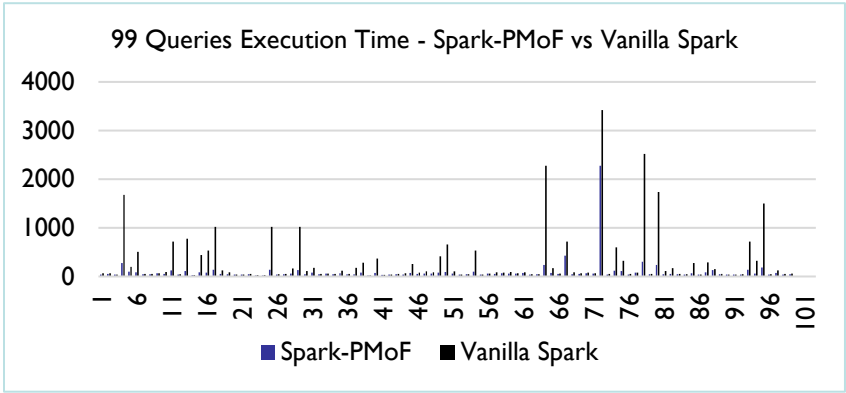
Spark PMoF: https://github.com/intel-bigdata/spark-pmof

Strata-ca-2019: https://conferences.oreilly.com/strata/strata-ca-2019/public/schedule/detail/72992

End-to-End Time Evaluation – TeraSort Workload

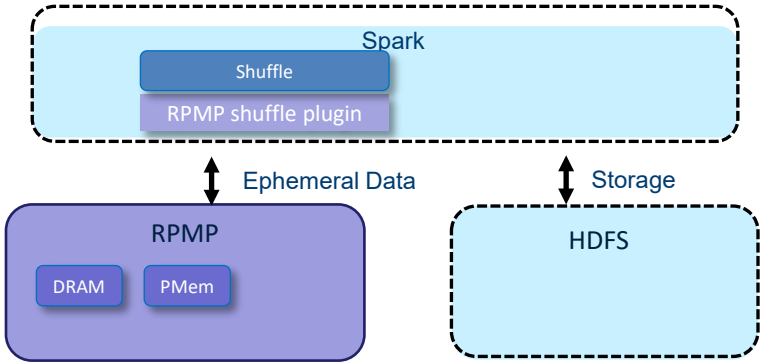
- Terasort
  - End-to-end time gains .vs Vanilla Spark: 22.7x
  - 1.29x speedup over 4x NVMe
  - PMoF shorten the remote read latency extremely
    - Readblocked time for HDD, NVMe & PMem (from Spark UI): 8.3min vs. 11s vs. 7ms
- PMem provides higher write/read bandwidth per node than HDD & NVMe and higher endurance
- Decision support workload
  - is less I/O intensive compared with Terasort
  - 3.2x speed up for total execution time of 99 queries
  - IO intensive workloads can be benefit more from PMoF performance improvement.





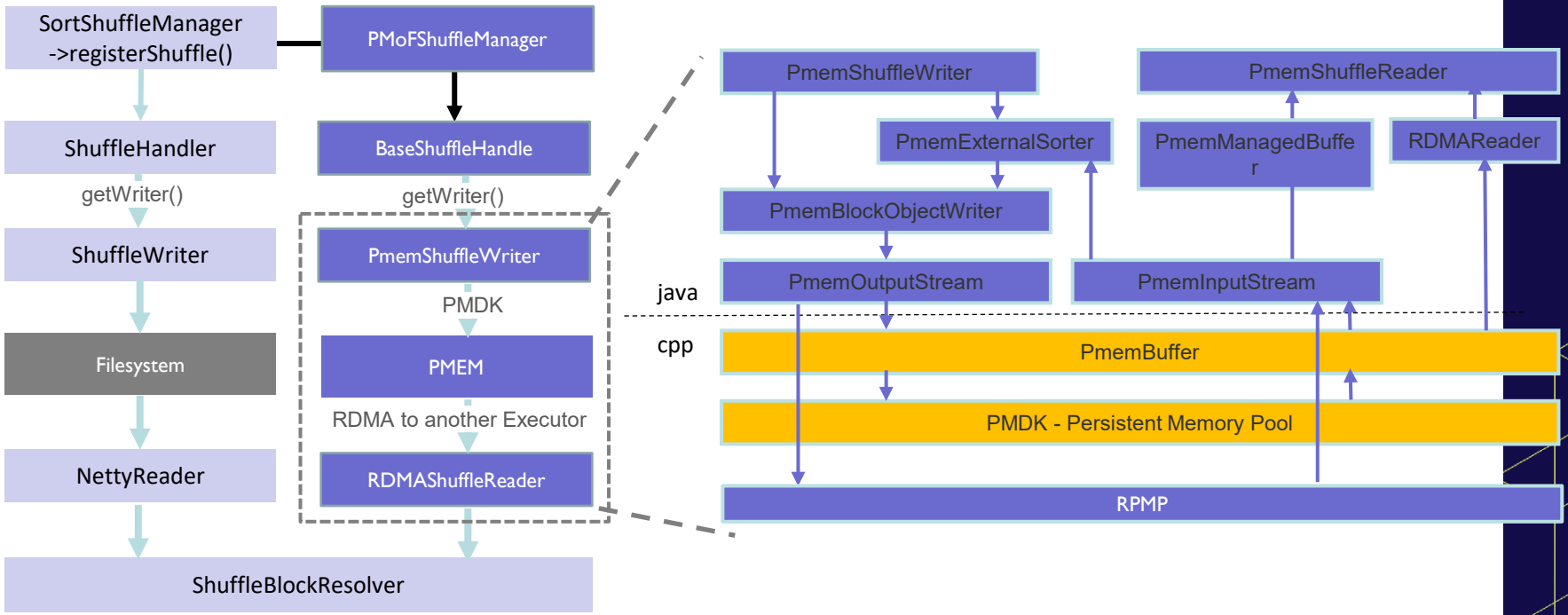
Performance results are based on testing as of 12/06/2019 and may not reflect all publicly available security updates. See configuration disclosure on slide for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. Configurations refer to page 39

RPMP as Spark disaggregated shuffle

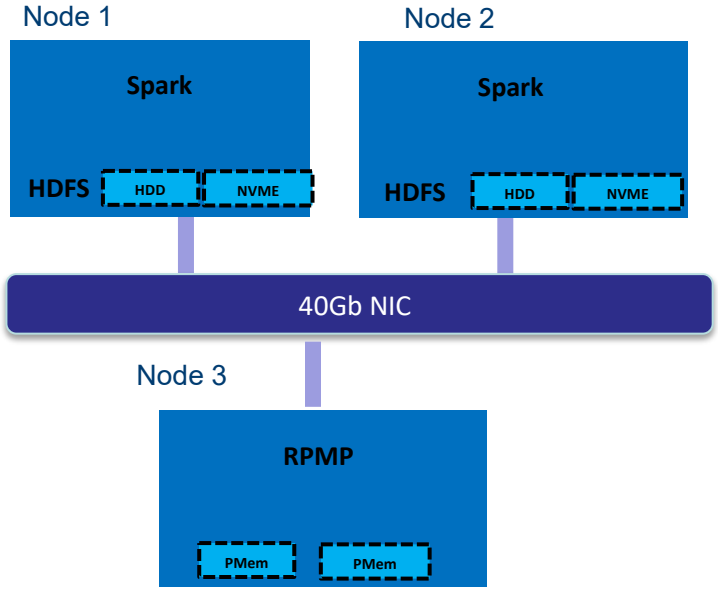


- RPMP as Spark disaggregated shuffle
  - A shuffle plugin build on top of RPMP clients
  - Shuffle I/O routes to RPMP
  - RPMP brings:
    - Decoupled shuffle I/O from a specific network/storage is capable of delivering dedicated SLA for critical applications
    - Fault tolerant in case on shuffle failure, no need for recompute
    - Reduced compute memory resource requirements
      - Offload spill as well, Balanced resource utilization
    - High performance & low latency leveraging state-of-art storage medium as storage media
      - To provide high performance, high endurance storage backend

RPMP integration to Spark Shuffle

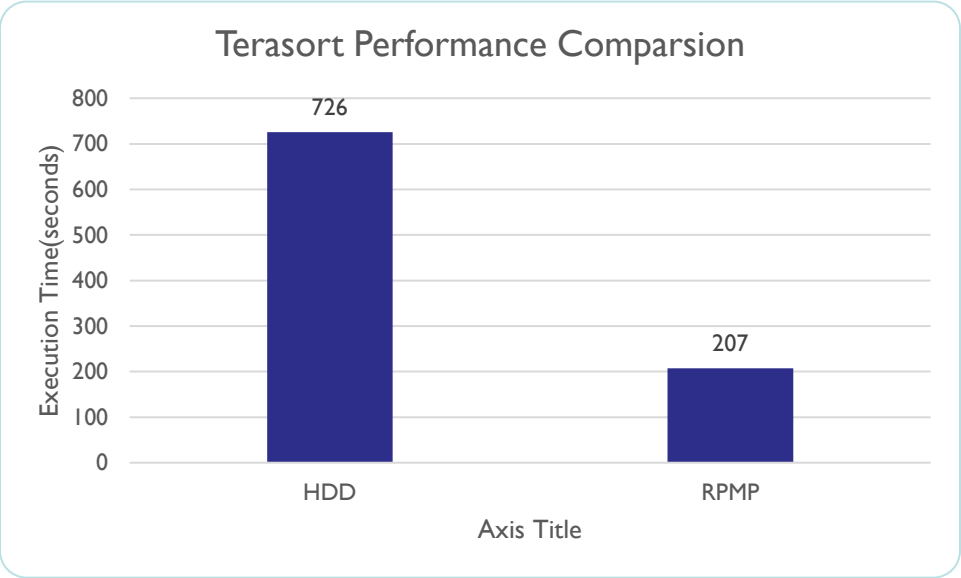


Performance Evaluation



- Configuration
  - Gold 6240 CPU @ 2.60GHz, 384GB Memory (12x 32GB 2666 MT/s)
  - 1x Mellanox ConnectX-4 40Gb NIC
  - Shuffle Devices：
    - 1x HDD for shuffle
    - RPMP w/ 8x 128GB Persistent Memory
    - Baseline: Shuffle on HDD
- Workload: Terasort 100GB

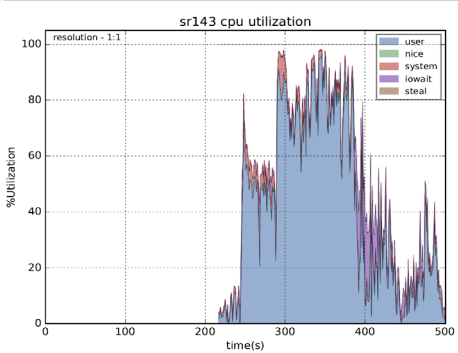
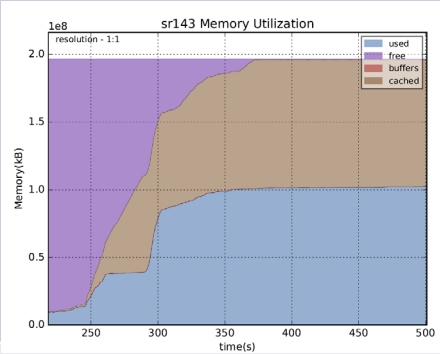
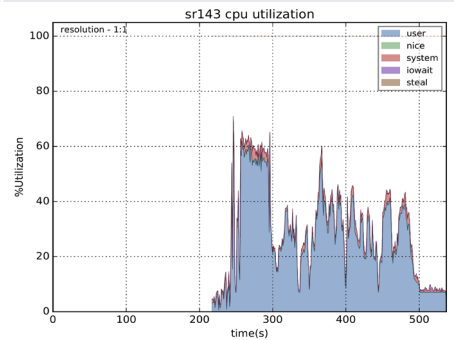
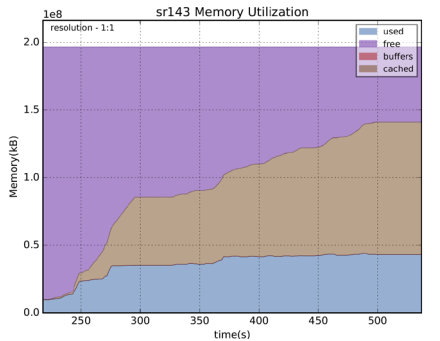
Performance Evaluation



- RPMP as shuffle delivers 3.5x performance speedup over Vanilla Spark

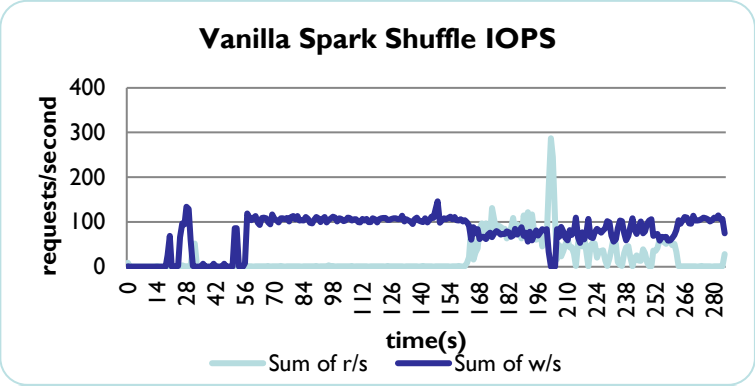
Performance results are based on testing as of 7/30/2020 and may not reflect all publicly available security updates. See configuration disclosure on slide for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. Configurations refer to page 40

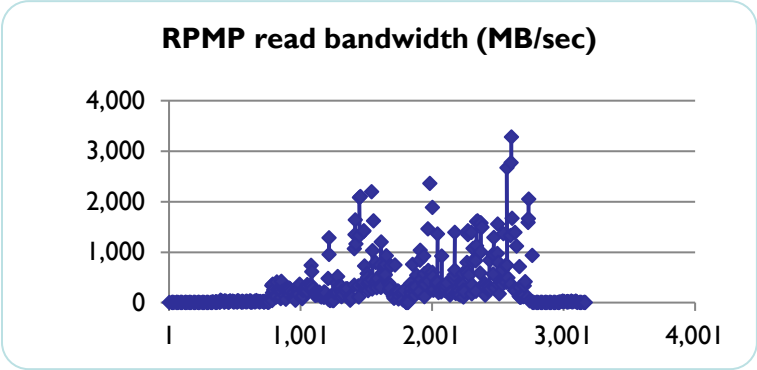
Performance Characteristics

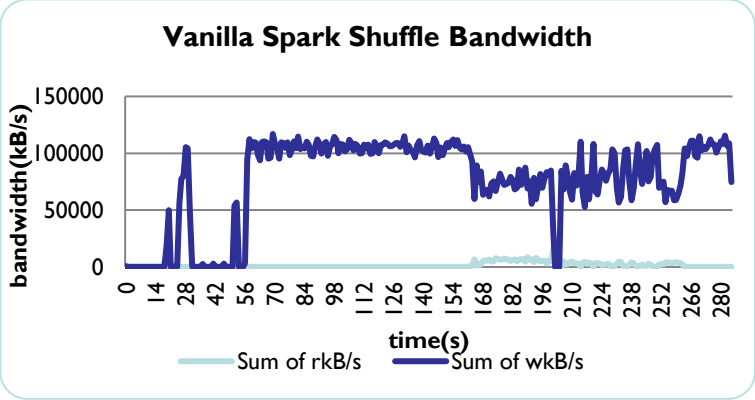
	CPU	Memory
Vanilla Spark		
RPMP		

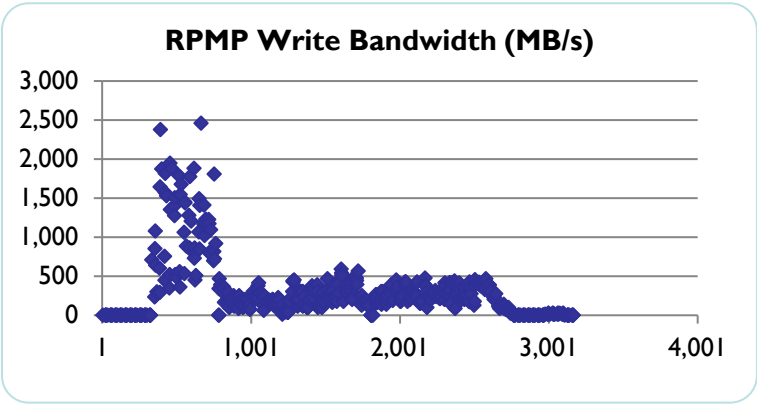
- RPMP improves compute node resolution utilization

Performance Characteristics









- RPMP eliminates Vanilla Spark's shuffle bottlenecks

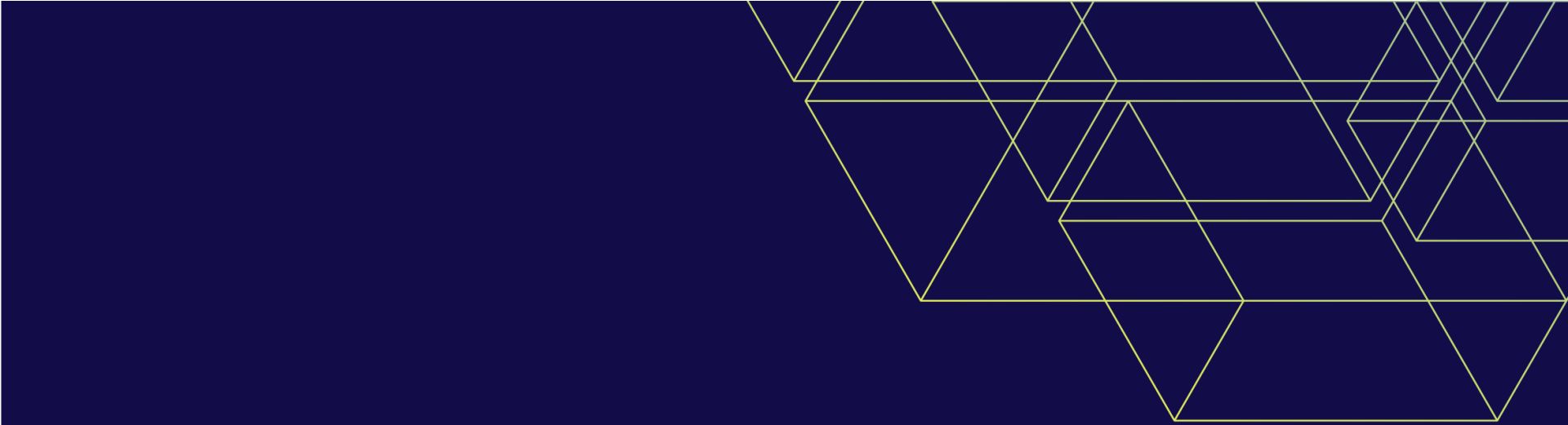
Summary

Summary

- Persistent Memory introduces a new high performance, cost efficient storage layer between DRAM and SSD
- Remote Persistent Memory extending PM new usage mode to new scenarios with RDMA being the most acceptable technology used for remote persistent memory access
- Remote Persistent Memory pool is a PMem based distributed storage system targeting as high-performance storage and resolve ephemeral data's performance issue
  - It is a distributed storage system build on top of PMem and powered by RDMA
  - Delivered rich API w/ user space access
  - Prototype in KV storage, disaggregated shuffle for Spark showed promising performance results

Credits

- This is a teamwork.
  - Thanks contributions of Xue Chendi and Ma Eugene.



Please take a moment
to rate this session.

Your feedback matters to us.

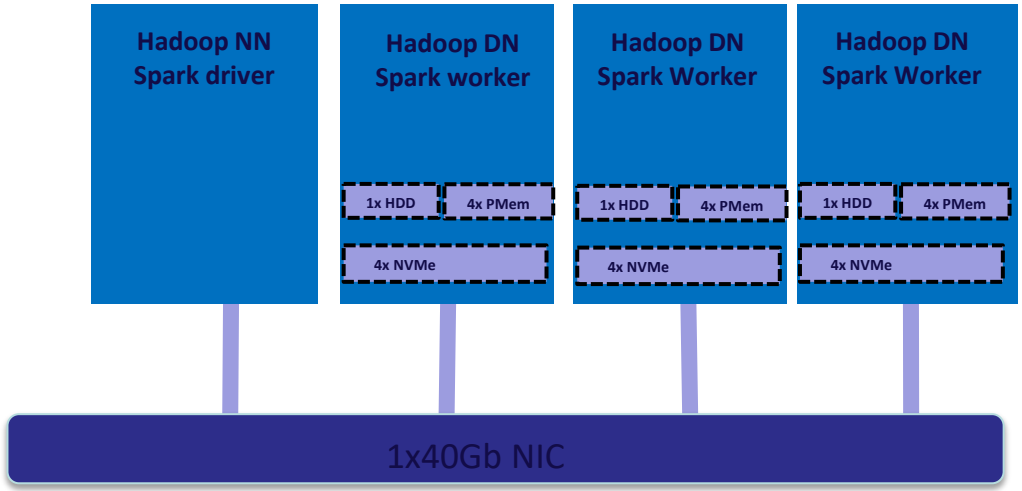
Legal Information: Benchmark and Performance Disclaimers

- Performance results are based on testing as of Feb. 2019 & Aug 2020 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.

Notices 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, Xeon, Optane, Optane Persistent Memory 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
- © Intel Corporation.

RPMem shuffle extension Benchmark configuration



4 Node cluster

Hardware:

- Intel® Xeon™ processor Gold 6240 CPU @ 2.60GHz, 384GB Memory (12x 32GB 2666 MT/s)
- 1x Mellanox ConnectX-4 40Gb NIC
- Shuffle Devices :
  - 1x HDD for shuffle
  - 4x 128GB Persistent Memory for shuffle
- 4x 1T NVMe for HDFS

Software:

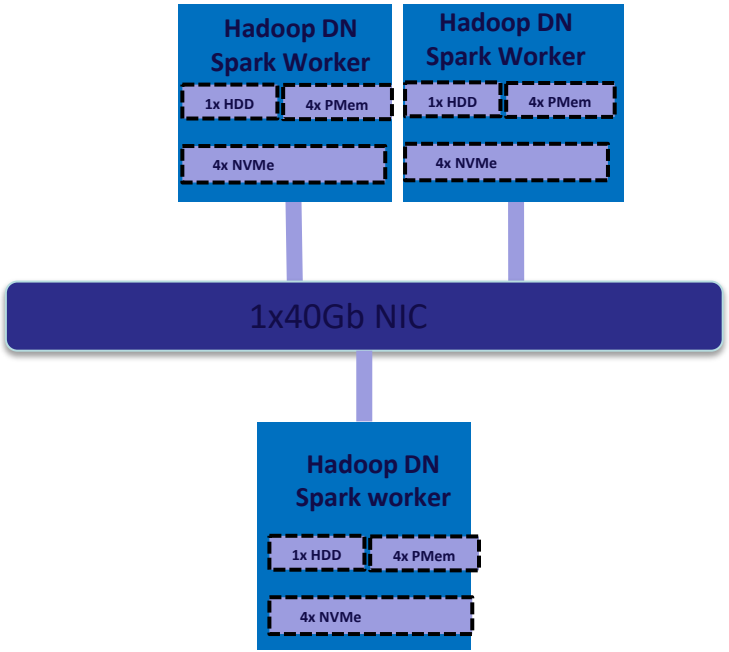
- Hadoop 2.7
- Spark 2.3
- Fedora 27

Workloads

Terasort 600GB

- 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

RPMP Benchmark configuration



3 Node cluster

Hardware:

- Intel® Xeon™ processor Gold 6240 CPU @ 2.60GHz, 384GB Memory (12x 32GB 2666 MT/s)
- 1x Mellanox ConnectX-4 40Gb NIC
- Shuffle Devices :
  - 1x HDD for shuffle
  - 8x 128GB Persistent Memory for shuffle
- 4x 1T NVMe for HDFS

Software:

- Hadoop 2.7
- Spark 2.3
- Fedora 27

Workloads

Terasort 600GB

- 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