LET'S GET OUR HANDS DIRTY: A COMPREHENSIVE EVALUATION OF DAQDB, KEY-VALUE STORE FOR PETASCALE HOT STORAGE

Presenter: Grzegorz Jereczek
on behalf of DAQDB team

CHEP 2019, Adelaide, Australia

Legal Notices & Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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

Intel, the Intel logo, Intel Xeon, Intel, Optane are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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

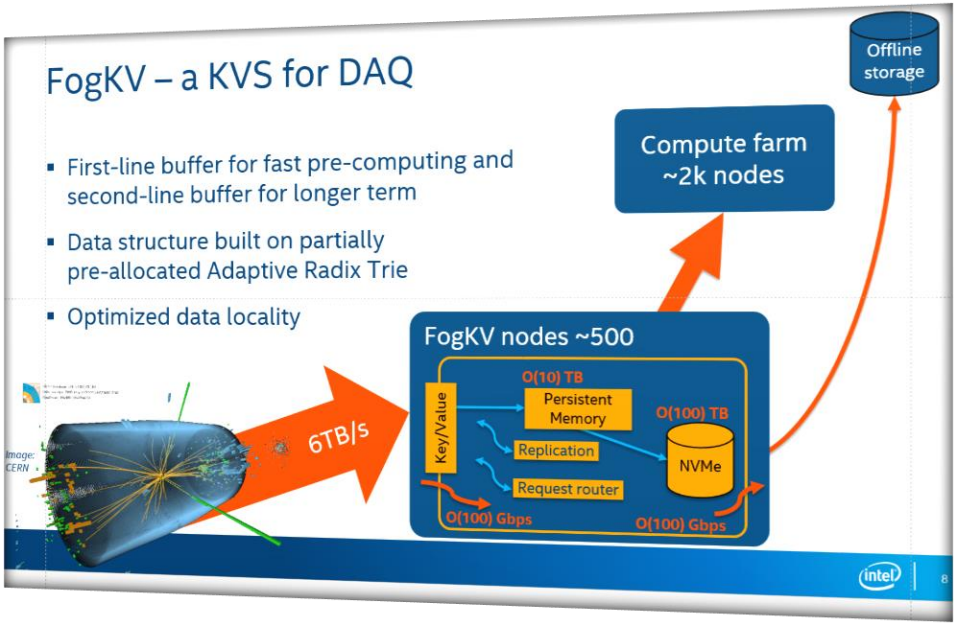
What has changed since CHEP'18?

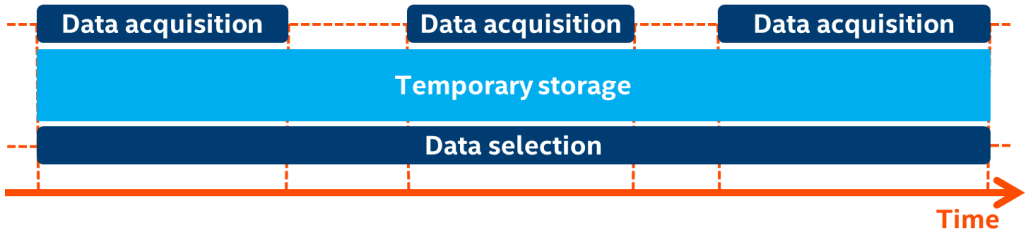
FogKV became DAQDB and is available @github: **github.com/daq-db/daqdb.**

Intel Optane DC Persistent Memory is now available.

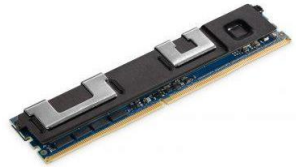
DAQDB has been integrated into ATLAS TDAQ framework.

Distributed mode has been enabled.





DAQDB highlights



Designed for Intel Optane DC Persistent Memory
providing data persistence with strong performance and affordable capacity.

Second-line NVMe-based storage to further extend the capacity.

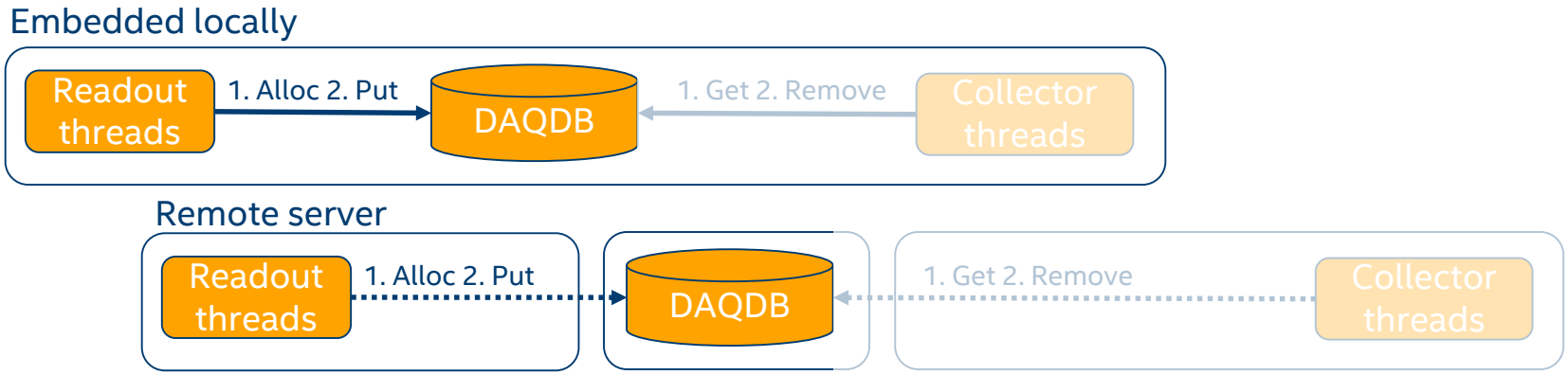
Data structure based on Adaptive Radix Trie (ARTree) for
efficient range queries.

DAQ-specific API featuring compound keys, range queries, and
next event retrieval.

DAQDB PERFORMANCE EVALUATION

Persistent data structure with volatile data in first-line buffer

Collection starting after readout



Server node:

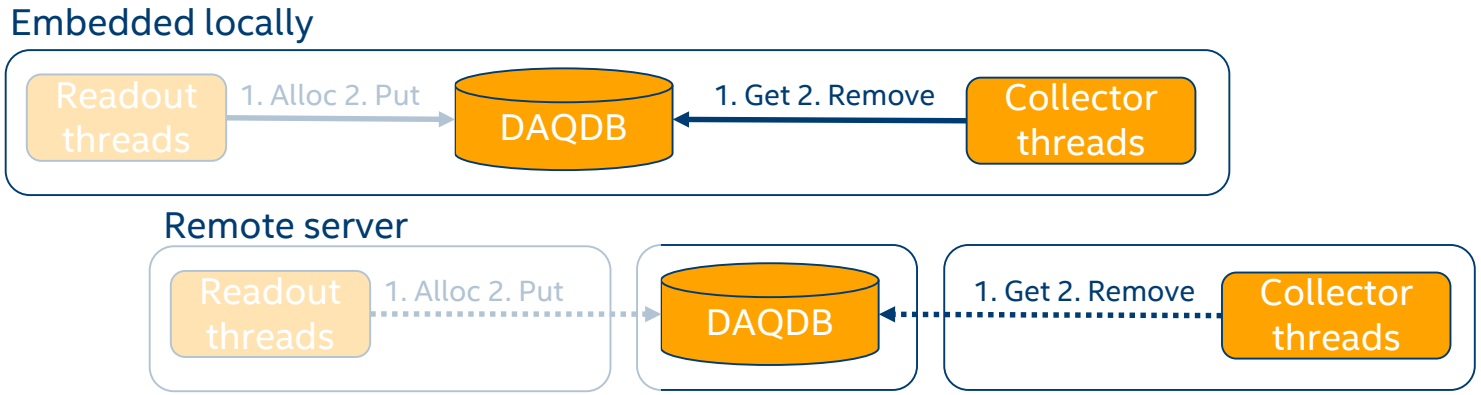
Intel(R) Xeon(R) Platinum 8280L CPU @2.70GHz,
6TB of Intel(R) Optane (TM) DC persistent memory
192GB of DDR4 DRAM

Client node:

Intel(R) Xeon(R) Gold 6252 CPU @2.10GHz

Centos 7.7, kernel 4.19, OFED 4.7
Mellanox ConnectX-5(R) with eRPC in raw Ethernet mode
Minidaq application
https://github.com/daq-db/daqdb/tree/master/apps/minidaq
https://erpc.io/

Collection starting after readout



Server node:

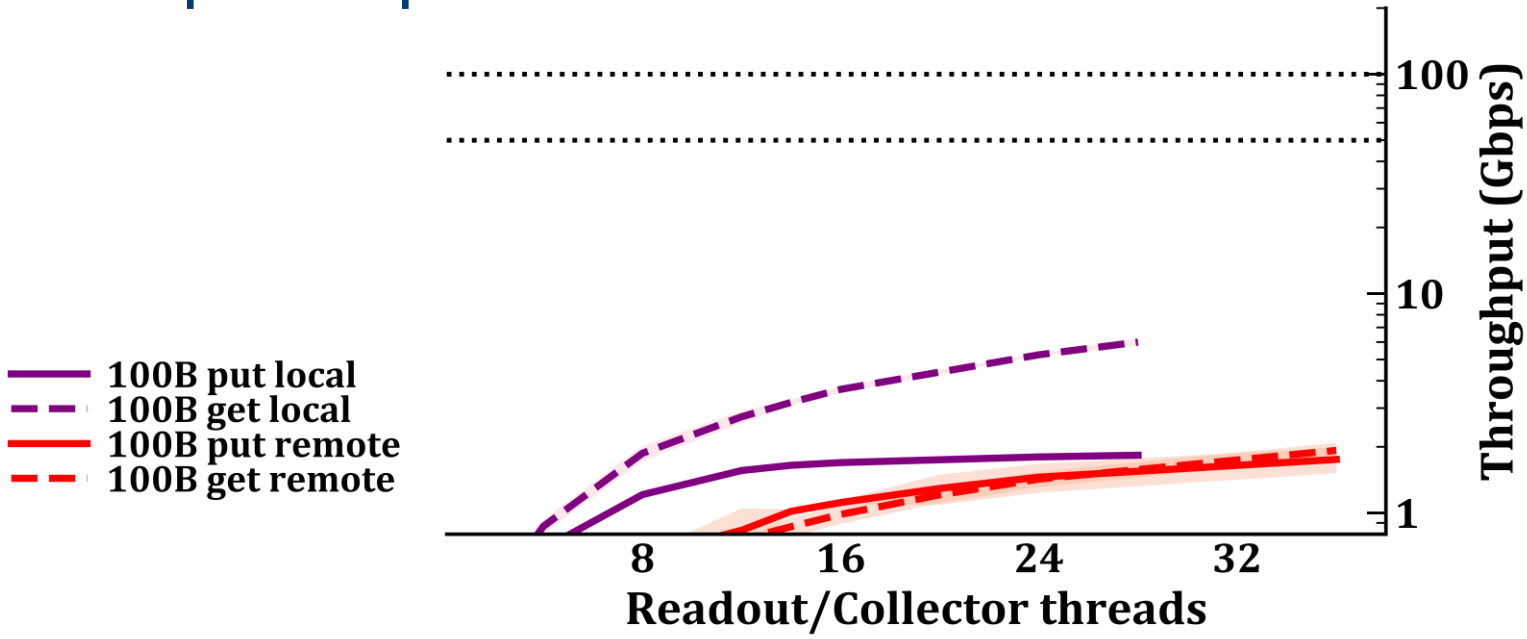
Intel(R) Xeon(R) Platinum 8280L CPU @2.70GHz,
6TB of Intel(R) Optane (TM) DC persistent memory
192GB of DDR4 DRAM

Client node:

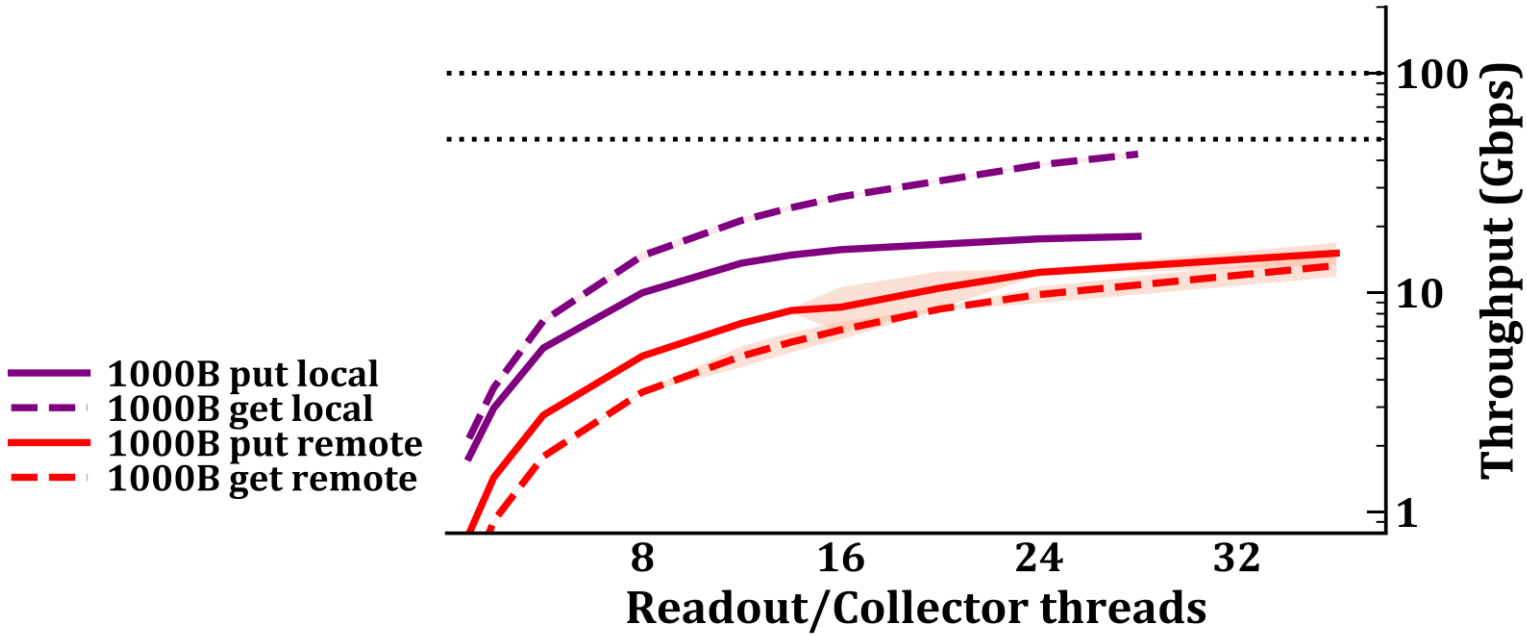
Intel(R) Xeon(R) Gold 6252 CPU @2.10GHz

Centos 7.7, kernel 4.19, OFED 4.7
Mellanox ConnectX-5(R) with eRPC in raw Ethernet mode
Minidaq application
https://github.com/daq-db/daqdb/tree/master/apps/minidaq
https://erpc.io/

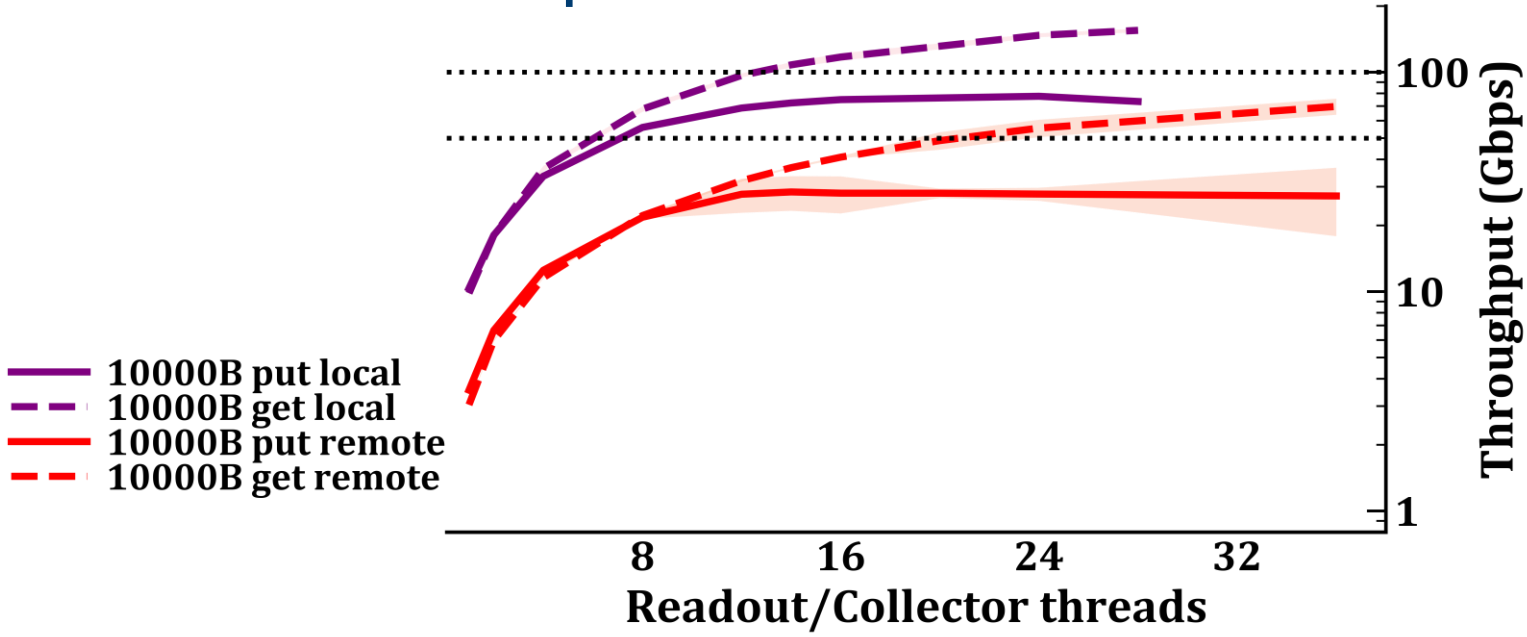
Good multi-core scalability while filling and emptying DAQDB with a single server CPU. Get requests profit from lockless access.



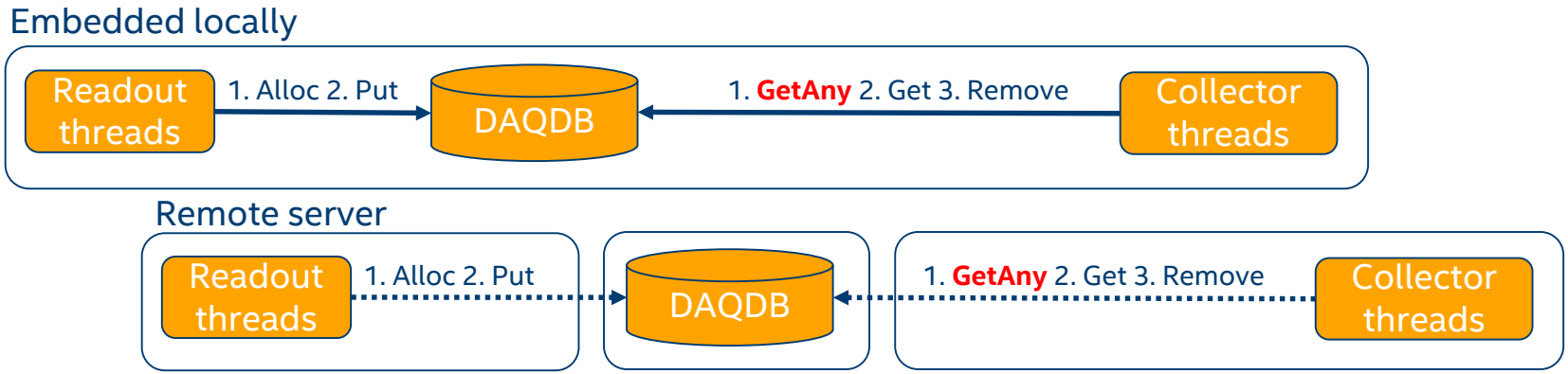
Server is CPU-bound, while client is latency-bound. eRPC adds segmentation latency with 1kB MTU.



At 10kB server is memory-bound. Memory and network IO add extra latency for the client. The collectors outperform now the readout.



Readout and collection in parallel



Server node:

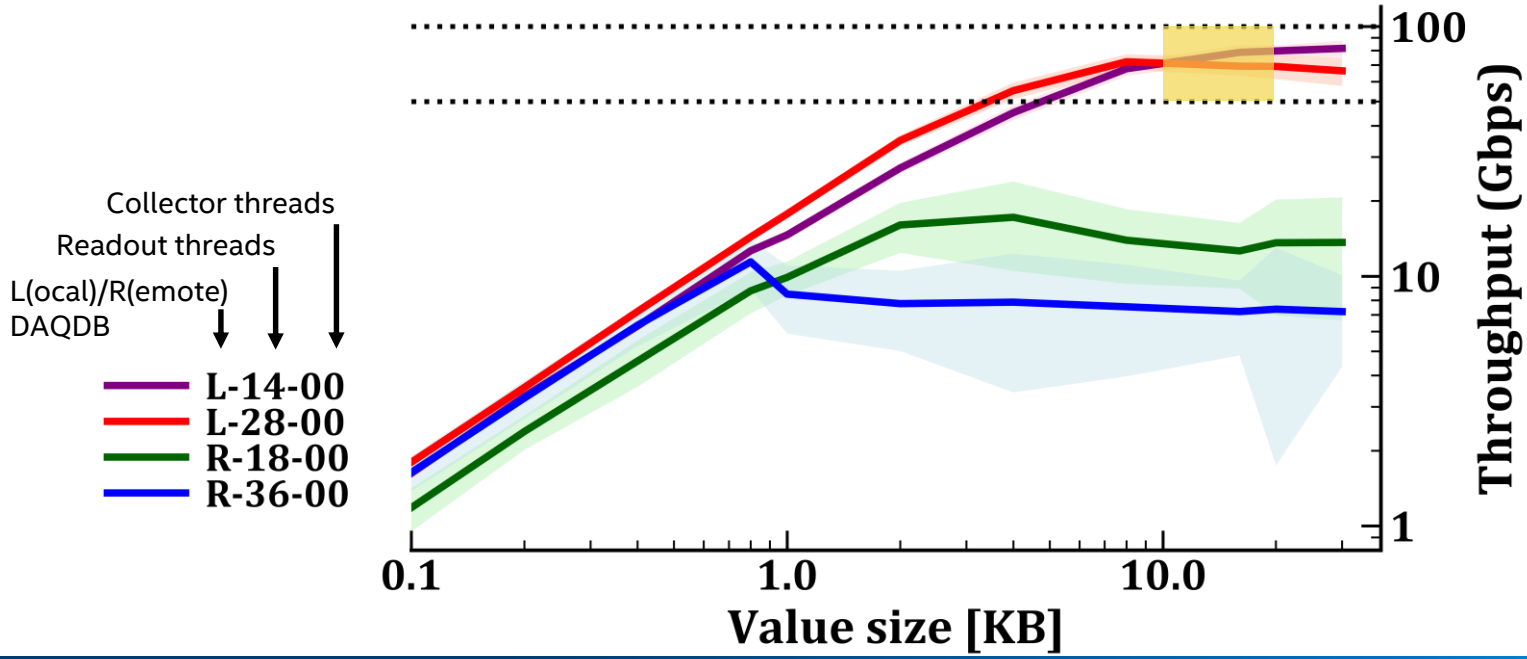
Intel(R) Xeon(R) Platinum 8280L CPU @2.70GHz,
6TB of Intel(R) Optane (TM) DC persistent memory
192GB of DDR4 DRAM

Client nodes:

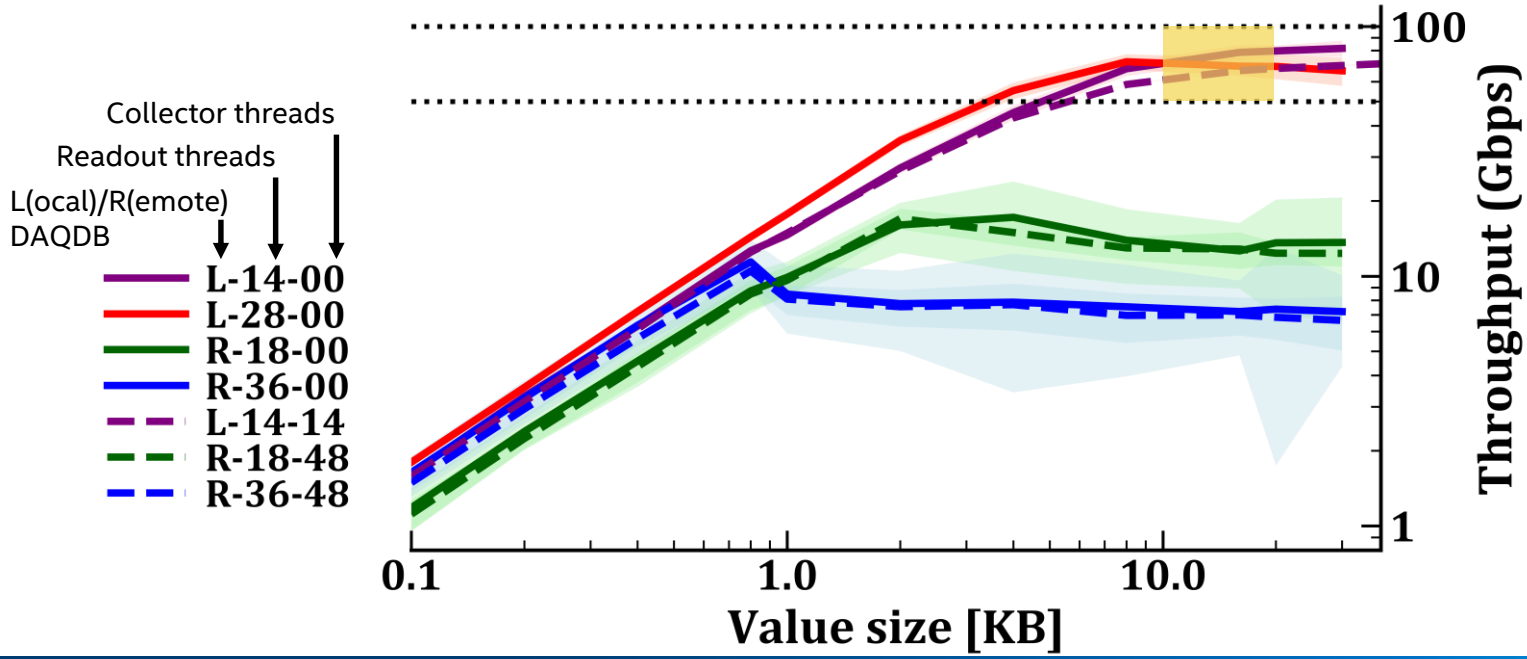
Intel(R) Xeon(R) Gold 6252 CPU @2.10GHz
Intel(R) Xeon(R) Gold 6140 CPU @2.30GHz

Centos 7.7, kernel 4.19, OFED 4.7
Mellanox ConnectX-5(R) with eRPC in raw Ethernet mode
Minidaq application
https://github.com/daq-db/daqdb/tree/master/apps/minidaq
https://erpc.io/

DAQDB is in the performance range required by ATLAS/CMS for local access, but requires optimization in remote mode.



Small impact on readout with data collection happening in parallel



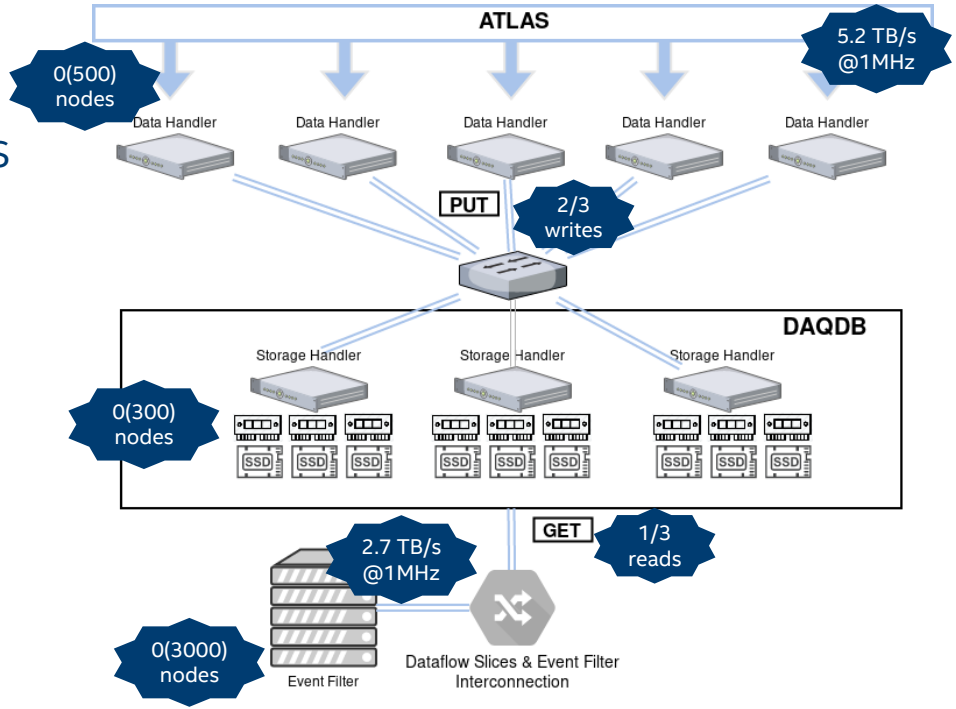
DAQDB being integrated with ATLAS TDAQ

Complete dataflow simulation:

- Writer application with embedded DAQDB.
- DAQDB client application for getting ATLAS fragments.

Next steps

- Optimization to reach target throughput.
- Standalone applications for fine-tuning.
- Multiple writer/reader applications.



SUMMARY & OUTLOOK

Lessons learned so far

Persistent memory proves to be a good candidate for lower-cost and higher-capacity buffering solution for DAQ than standard DRAM.

Initial performance evaluation of DAQDB gives promising results as for a generic KVS solution for DAQ.

GetAny as first DAQ-oriented feature is functional.

ARTree has non-negligible performance/memory overhead and requires more effort in implementation. Simpler data structures might be good enough.

Outlook

Implementation and evaluation of second-line buffer.

Optimization of distributed mode of operation, move to asynchronous mode.

Alternative data structures.

Multi-node scaling.

A JOINT WORK ON A GENERIC KVS FOR DAQ

Looking for contributors!

https://github.com/daq-db/daqdb

ARTree 256-element node lock is the bottleneck @100B value size

```
--82.44%--DaqDB::MinidaqRoNode::_Task
          |
          |--77.00%--DaqDB::KVStore::Alloc
          |          |
          |           --76.92%--DaqDB::ARTree::AllocValueForKey
          |                     |
          |                      --70.56%--DaqDB::TreeImpl::findValueInNode
          |                                |
          |                                |--32.80%--pmemobj_mutex_lock
          |                                |
          |                                 --27.52%--pmemobj_mutex_unlock
          |
           --5.00%--DaqDB::KVStore::Put
```

Memory IO is the bottleneck @10KB

```
--93.78%--DaqDB::MinidaqRoNode::_Task
          |
          |--59.54%--pmem_memcpy_nodrain
          |
          |--20.13%--DaqDB::KVStore::Alloc
          |          |
          |           --20.11%--DaqDB::ARTree::AllocValueForKey
          |                     |
          |                     |--13.11%--DaqDB::TreeImpl::findValueInNode
          |                     |          |
          |                     |           --5.65%--pmemobj_publish
          |                     |
          |                      --5.98%--pmemobj_xreserve
          |
           --13.48%--DaqDB::KVStore::Put
```