VeloC-SZ Project

Franck Cappello: lead PI
Argonne
National Laboratory

Kathryn Mohror, co-PI
Lawrence Livermore
National Laboratory

- **VeloC**: Very Low Overhead Transparent Multilevel Checkpoint/Restart

  Bogdan Nicolae: ANL

- **SZ**: Fast, Effective, Parallel Error-bounded Exascale Lossy Compression for Scientific Data

  Sheng Di: ANL

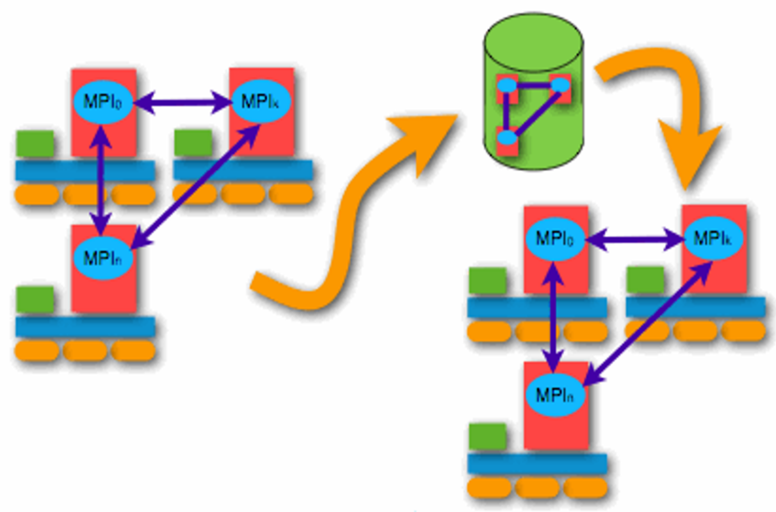
VELOC: Very Low Overhead Checkpointing System

Bogdan Nicolae, Greg Kosinovsky, Adam Moody, Kathryn Mohror, Franck Cappello

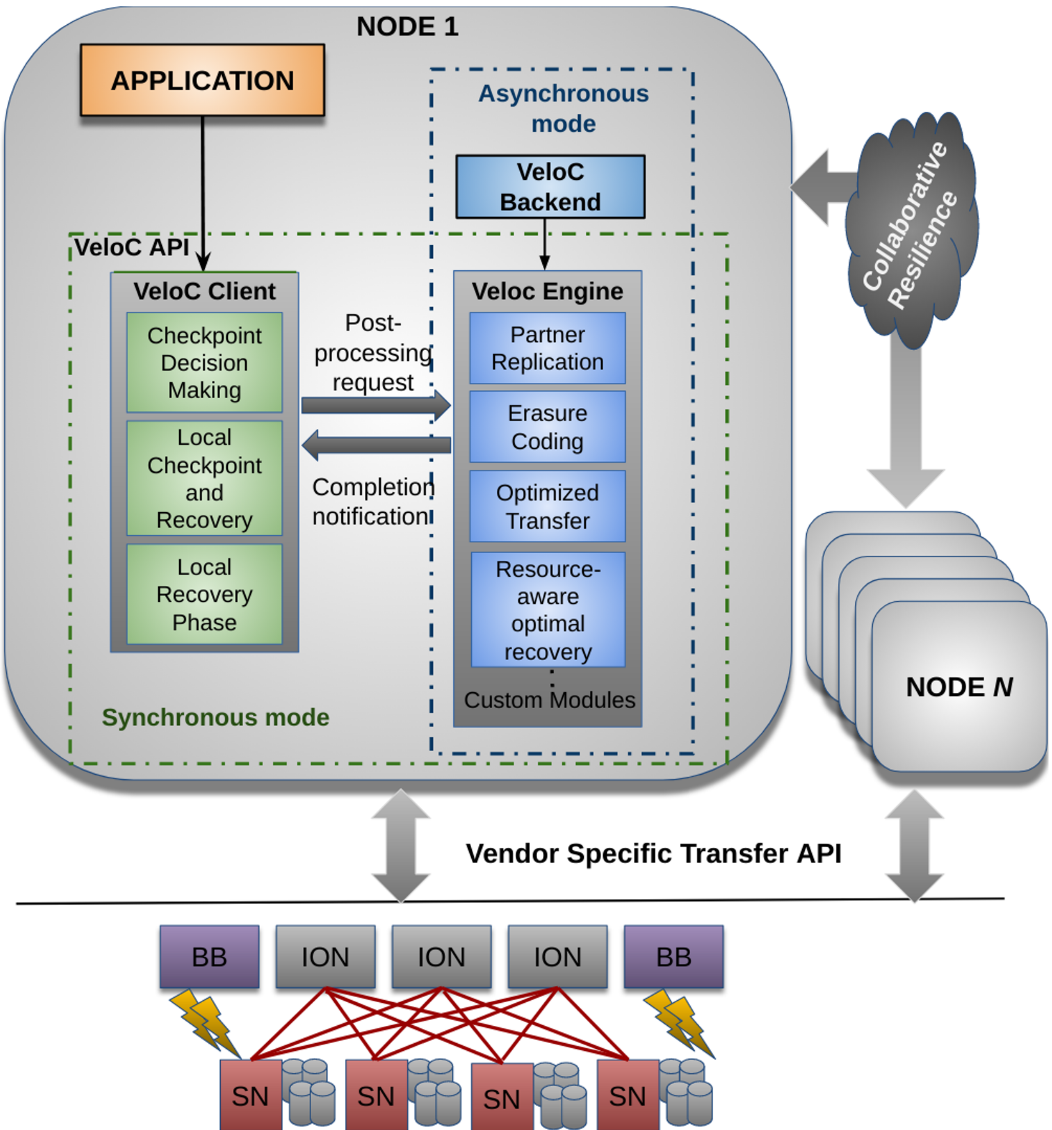
Argonne National Laboratory
Lawrence Livermore National Laboratory

Use Cases of Checkpointing

- **Defensive:**
  - Fault tolerance based on checkpoint-restart
- **Administrative:**
  - Suspend-resume (e.g. make room for higher priority jobs)
  - Migration
  - Debugging
- **Productive:**
  - Share and reuse datasets between workflow tasks (e.g., simulation + analytics)
  - Revisit previous intermediate datasets (e.g. adjoint computations)
  - Provenance tracking



VELOC: Overview



- High Performance and Scalability
- Hides complexity of interaction with deep storage stacks
- Configurable multi-level resilience:
  - L1: Local write
  - L2: Partner replication, XOR encoding, RS encoding
  - L3: Optimized transfer to external storage
- Configurable mode of operation:
  - Synchronous mode: resilience engine runs in application process
  - Asynchronous mode: resilience engine in separate backend process (VeloC does not die if app dies due to software failures)
- Easily extensible:
  - Custom modules can be added for additional post-processing in the engine (e.g. compression)

*Web: **https://veloc.readthedocs.io***

Lightweight Memory-Based API

Before:

```
MPI_Init(&argc, &argv);
// further initialization code
// allocate two critical double arrays of size M
h = (double *) malloc(sizeof(double *) * M * nbLines);
g = (double *) malloc(sizeof(double *) * M * nbLines);
// set the number of iterations to 0
i = 0;
while (i < n) {
    // iteratively compute the heat distribution
    // increment the number of iterations
    i++;
}
MPI_Finalize();
```

- **Critical state:** Dynamically protect and unprotect, checkpoint later - no need to remember (or have a pointer to) all required data structures at checkpoint time
- **Convenience:** No need to worry about how to serialize critical data structures
- **Performance:** VELOC decides when and how to serialize

After:

```
MPI_Init(&argc, &argv);
VELOC_Init(MPI_COMM_WORLD, argv[2]); // (1): init
// further initialization code
// allocate two critical double arrays of size M
h = (double *) malloc(sizeof(double *) * M * nbLines);
g = (double *) malloc(sizeof(double *) * M * nbLines);
// (2): protect
VELOC_Mem_protect(0, &i, 1, sizeof(int));
VELOC_Mem_protect(1, h, M * nbLines, sizeof(double));
VELOC_Mem_protect(2, g, M * nbLines, sizeof(double));
// (3): check for previous checkpoint version
int v = VELOC_Restart_test("heatdis", 0);
// (4): restore memory content if previous version found
if (v > 0) {
    printf("Previous checkpoint found at iteration %d, initiating r
    // v can be any version, independent of what VELOC_Restart_test
    assert(VELOC_Restart("heatdis", v) == VELOC_SUCCESS);
 } else
     i = 0;
 while (i < n) {
     // iteratively compute the heat distribution
     // (5): checkpoint every K iterations
     if (i % K == 0)
         assert(VELOC_Checkpoint("heatdis", i) == VELOC_SUCCESS);
      // increment the number of iterations
      i++;
 }
 VELOC_Finalize(0); // (6): finalize
 MPI_Finalize();
```

Apps: ECP ExaSky

Use case: Multi-iteration computation checkpointing particles

Integration goals:

- Isolate checkpointing code in a single place for easier maintenance
- Easy way to switch checkpointing on/off
- Modular architecture to share critical data used for checkpoints with other in-situ plugins (e.g., analytics, post-processing, etc.)

Progress since last year:

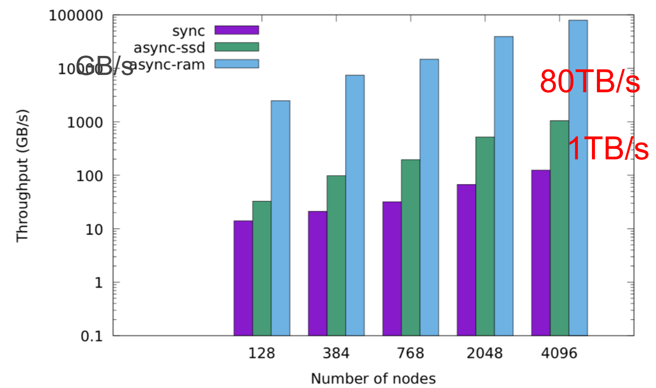
- New VELOC control plane to minimize dependencies on external libraries (previously based on Boost, now UNIX sockets alternative available)
- New deployment model to simplify running HACC with VELOC (eliminates wrapper scripts and the need to launch the backend beforehand)
- Refactored VELOC plugin to minimize initialization overhead (since backend is launched by VELOC at initialization)
- Demonstrated performance and scalability at full scale on Summit compared with GIO (optimized blocking I/O library)

Next steps:

- Integration with object-based storage (DAOS)
- We have a draft paper discussing the root causes (CPU, Memory, different I/O strategies) of the interferences, which is the starting

Large-scale performance projection for HACC from 128 and 768 node measurements on Theta (Lustre max BW: 210 GB/s)

Note*: checkpointing 1 file per process is embarrassingly parallel in L1 with VeloC



Summit: 1024 and 4096 nodes (¼ and full machine size) using 6144 and 24576 GPUs (HACC team provided use cases)

Approach VeloC L1: Ram	**Ckpt overhead (blocking)**	**Async interference (next step slowdown)**
GIO: 1024 nodes	11.2s (539 GB/s)	N/A
VELOC: 1024 nodes	0.5s (67 TB/s)	5s → needs interference mitigation
GIO: 4096 nodes	90s (200GB/s)	N/A
VELOC: 4096 nodes **FOM* run**	0.081 (224 TB/s)	6s→ needs interference mitigation

*FOM: Figure Of Merit

EXASCALE COMPUTING PROJECT

Apps: ECP LatticeQCD

Use case:

- Checkpoint Gauge Field Vector and RNG at end of each trajectory

Integration Goal:

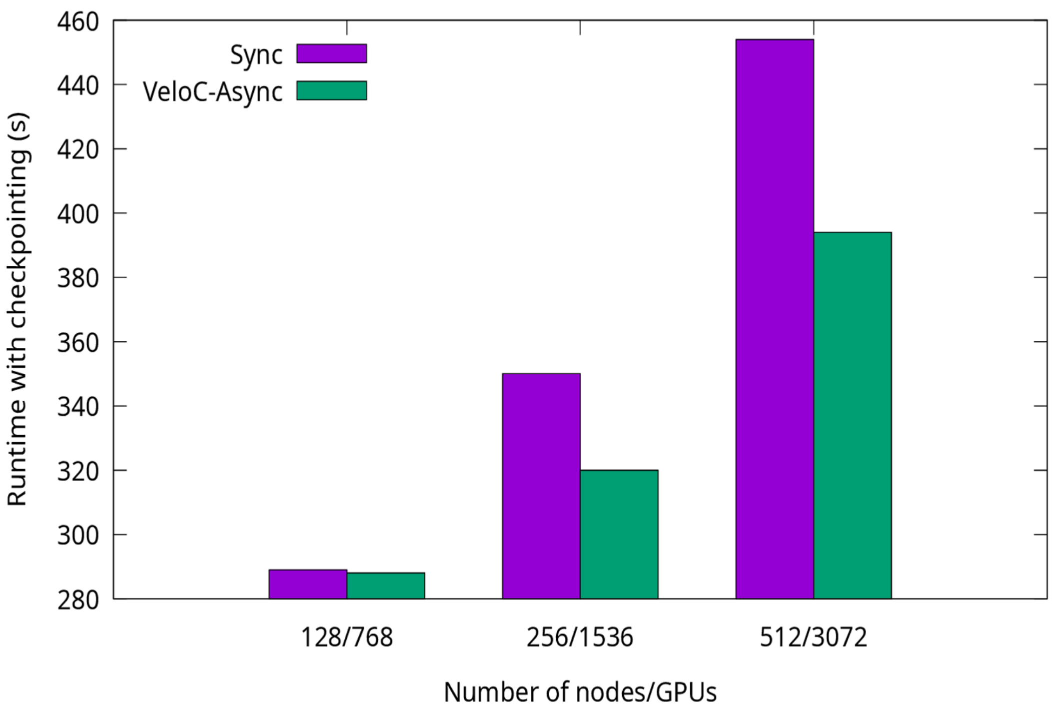
- Use VeloC within the CPS (Columbia Physics System, C++):
- Make it available as a branch of the CPS code

Solution:

- One contiguous memory region holds the majority of critical data and needs to be protected, which warrants the use of the memory-based mode
- Currently ignoring some smaller data structures, for which original checkpointing mechanism is used
- Integration performed by CPS team

Next Steps:

- Leverage native serialization support for complex C++ data structures
- This will enable full checkpointing support with VELOC



Results:

- Theta: small-scale run on 2, 4, 8 nodes
  - Benefits: up to 6x reduction of ckpt overhead
- Summit: large-scale run on 128, 256, 512 nodes
  - Benefits: better scalability (2x more nodes => 2x increase in difference between sync and async)

Apps: ECP EXAALT

Use case: Master-worker model

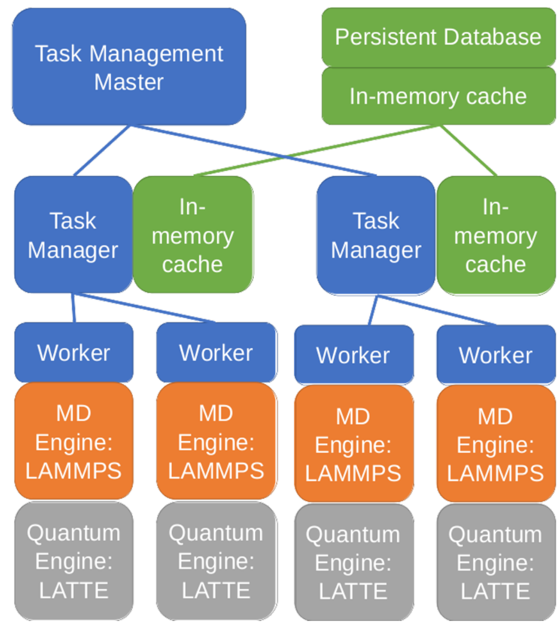
- Resilience: master needs to checkpoint one file
- Checkpoint: 1-10 GB, highly irregular, needs serialization of complex data structures, tends to increase with number of workers and time

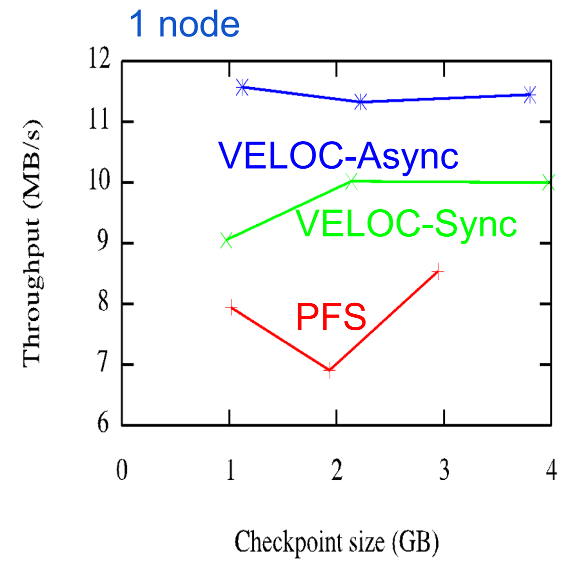
Integration Goal:

- Seamless transition from custom checkpointing mechanism to VeloC
- Add transparent support for asynchronous checkpointing

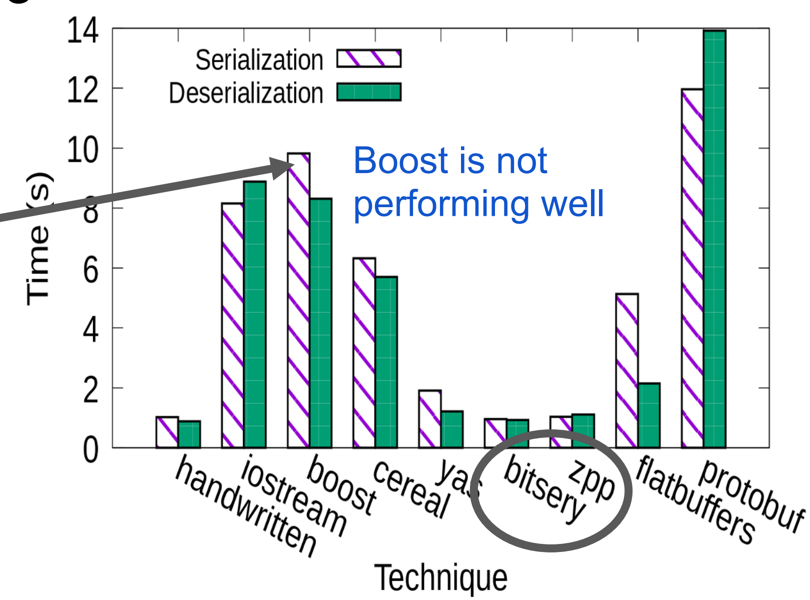
Work in progress:

- Currently relies on existing serialization mechanisms (Boost)
- VELOC goal: Improve serialization performance
- Switch to memory-based mode





Serialization is a significant bottleneck (low throughput per node), reduces effectiveness of async I/O



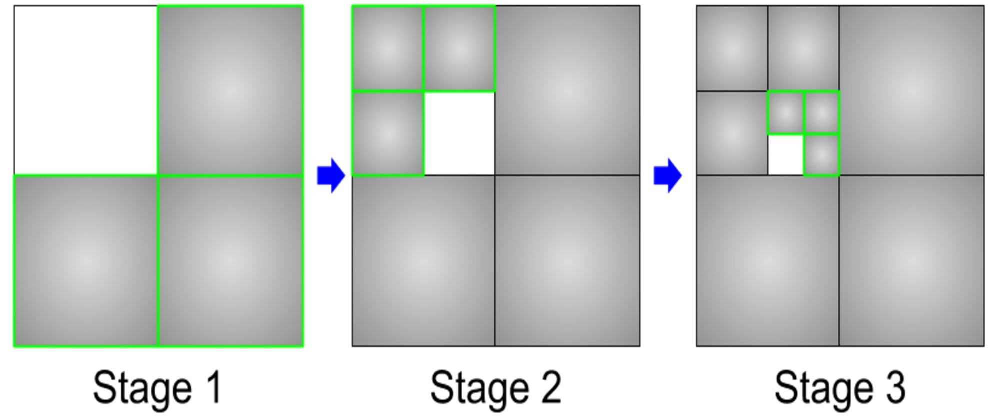
Comparison of serialization techniques:

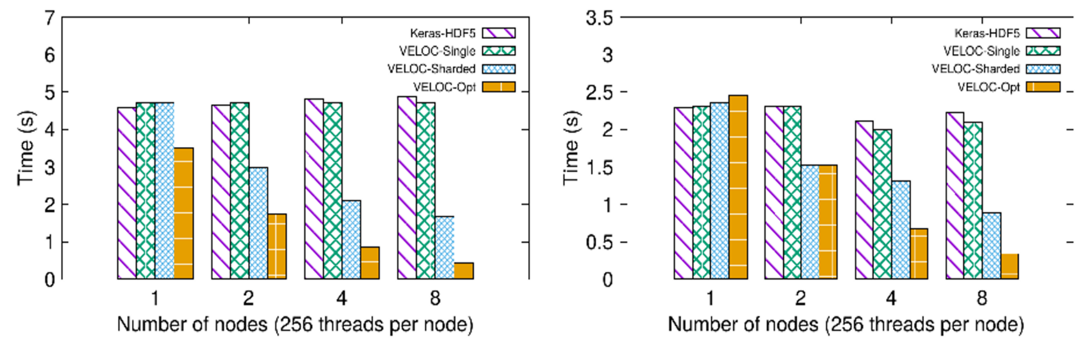
- New modern C++ libraries have been proposed recently
- Up to an order of magnitude improvement
- VELOC has generic support (mix and match any libraries as desired)

EXASCALE COMPUTING PROJECT

Apps: ECP CANDLE

- CANDLE: Cancer Deep Learning Framework
- Pattern relevant for checkpointing:
  - Split up the training data into subsets, iteratively train on most remaining subsets
  - Weight sharing from one subset to the next (incremental learning)
  - Multiple variations with common ancestors: need to efficiently replicate a partially trained model
  - Allows for investigations into data quality and learning patterns
- Runs at large scale on the Summit supercomputer
- Solution: Low-overhead checkpointing and cloning of partially trained models





(b) Blocking phase (lower is better). (c) Runtime overhead (lower is better).

Results: 5x-10x less blocking and runtime overhead, scales well for data-parallel training

More info: J. Wozniak, H. Yoo, J. Mohd-Yusof, B. Nicolae, N. Collier, J. Ozik, T Brettin, R. Stevens. 2020. High-bypass Learning: Automated Detection of Tumor Cells That Significantly Impact Drug Response. In MLHPC'20: 6th Workshop on Machine Learning in HPC Environments (held in conjunction with SC 2020).

EXASCALE COMPUTING PROJECT

Thanks

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Web: https://veloc.readthedocs.io