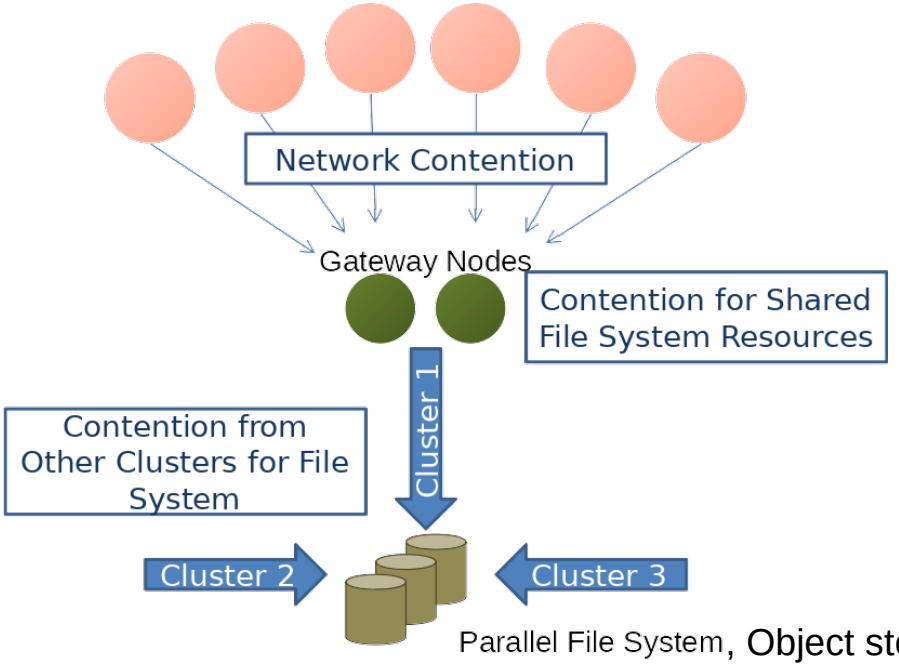
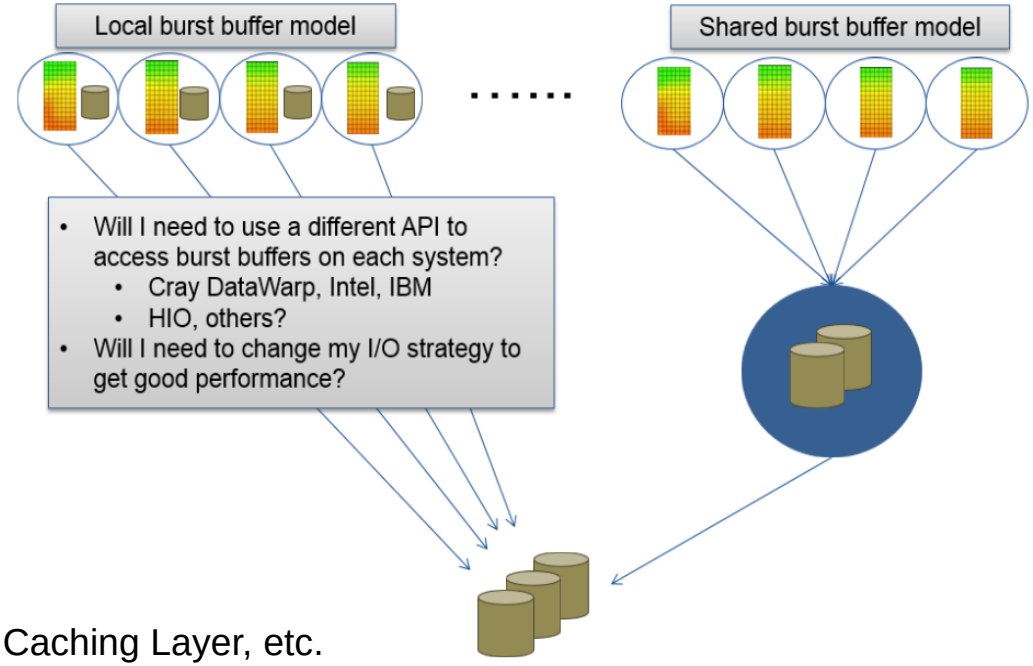
Beyond Fault Tolerance: Use Cases of Checkpoint-Restart in HPC

Bogdan Nicolae
Argonne National Laboratory

Exascale Checkpoint-Restart

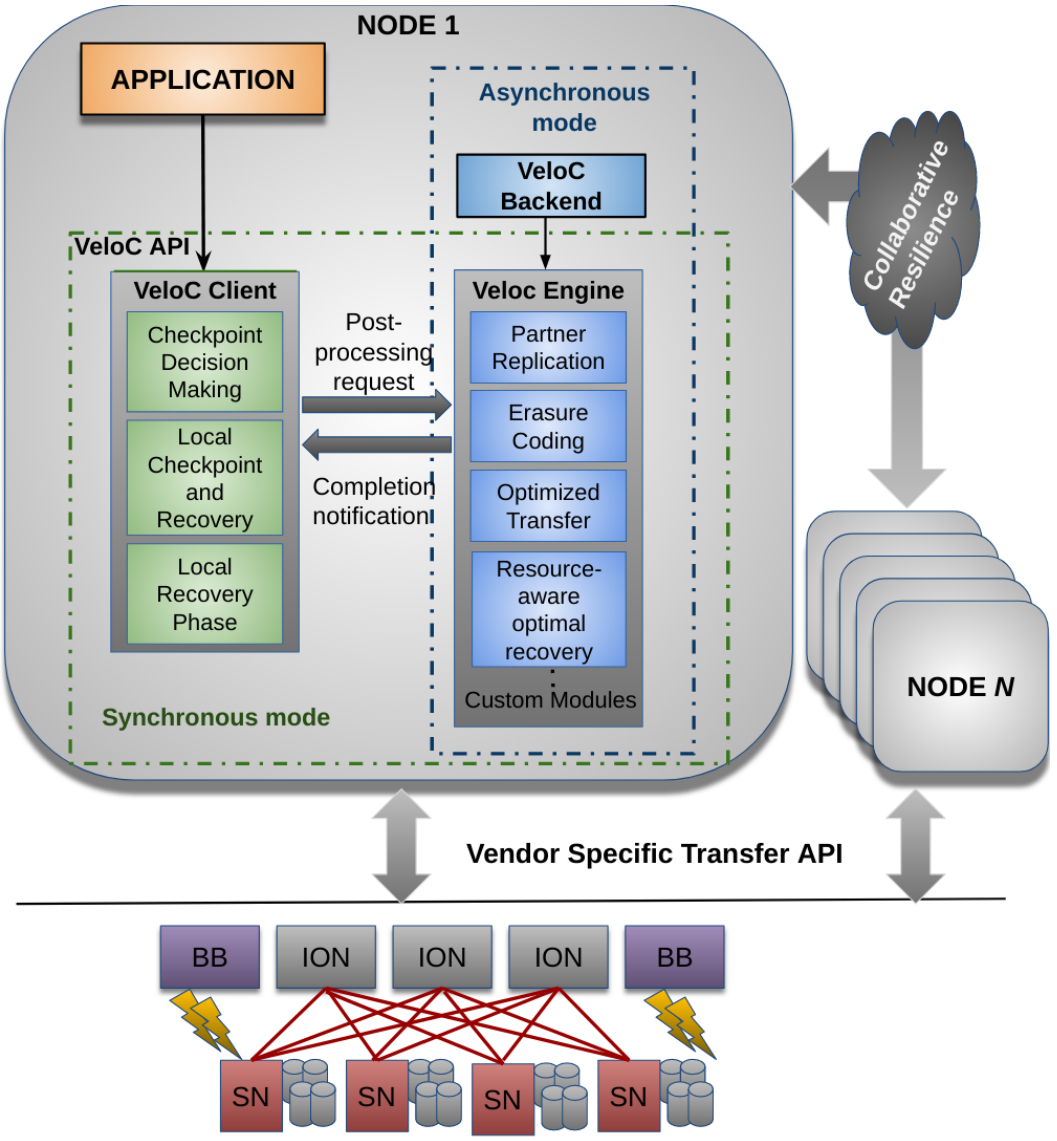
- Main HPC resilience solution
- Two major challenges:





- Performance and scalability
  - More failures at Exascale implies need to checkpoint more frequently
  - Less I/O bandwidth available per processing element
- Heterogeneity and complexity of storage hierarchy
  - Many options in addition to PFS: burst buffers, object stores, caching layers, etc.
  - Many vendors, each with its own API

VeloC: Very Low Overhead Checkpointing

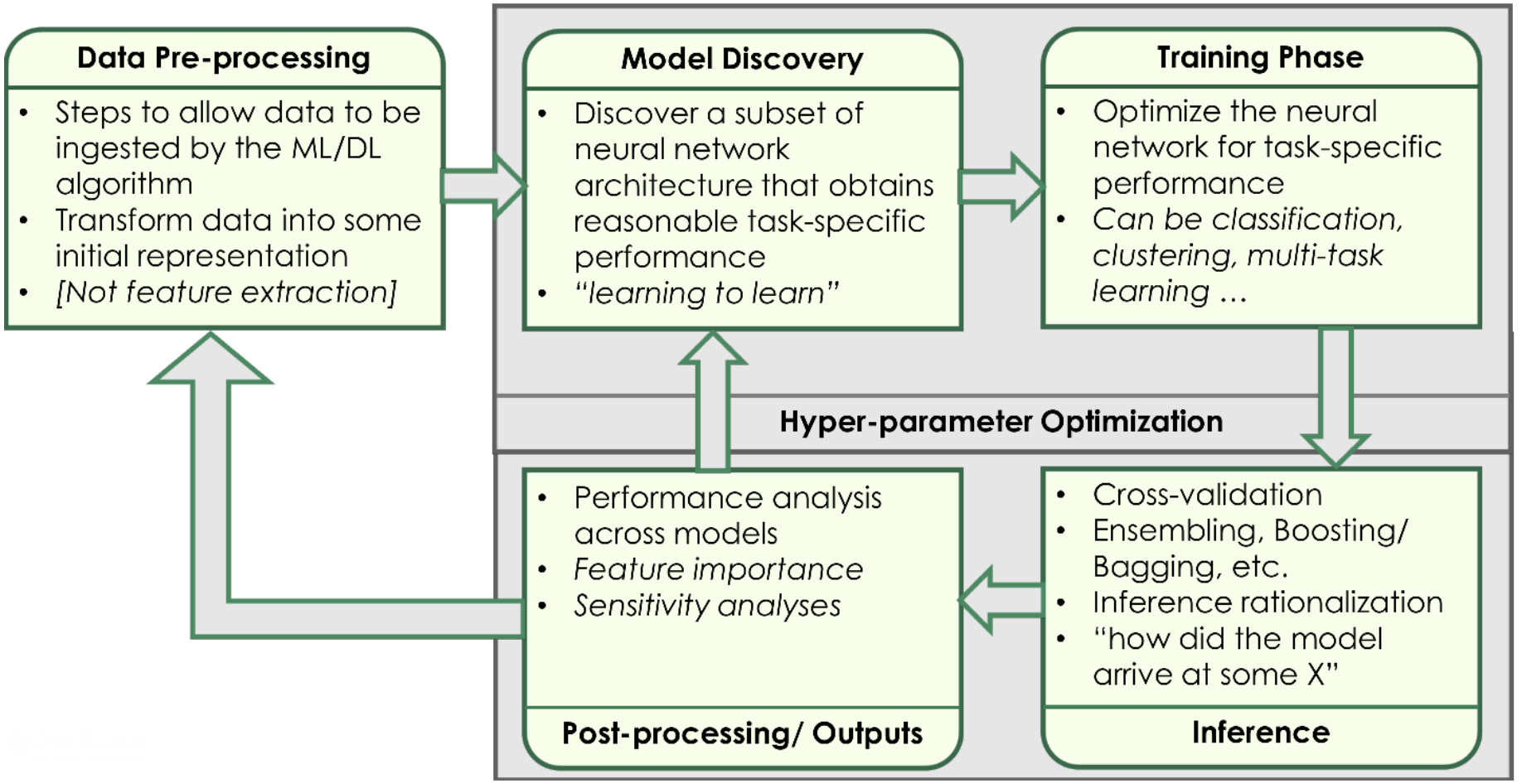


- High Performance and Scalability
- Hides complexity of interaction with deep storage stacks
- Configurable resilience strategy:
  - L1: Local write
  - L2: Partner replication, XOR encoding, RS encoding
  - L3: Optimized transfer to external storage
- Configurable mode of operation:
  - Synchronous 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)

HPC Suspend-Resume Support to Mix Batch with Real-Time Jobs

- Need to make room on for real-time jobs by suspending HPC batch jobs and resuming them later
  - Example: Experimental steering
- Problem: Given a maximum delay T to act on data, how to checkpoint and replace the batch job with the real-time job to satisfy deadline T with minimal loss of progress
- Challenges:
  - Simple solution: suspend as fast as possible, the load real-time job
    - Can we do better (e.g. suspend and load concurrenly)?
  - Is transparent checkpointing at system level feasible?
  - Any special memory and I/O patterns that can be exploited?
  - We are looking for applications

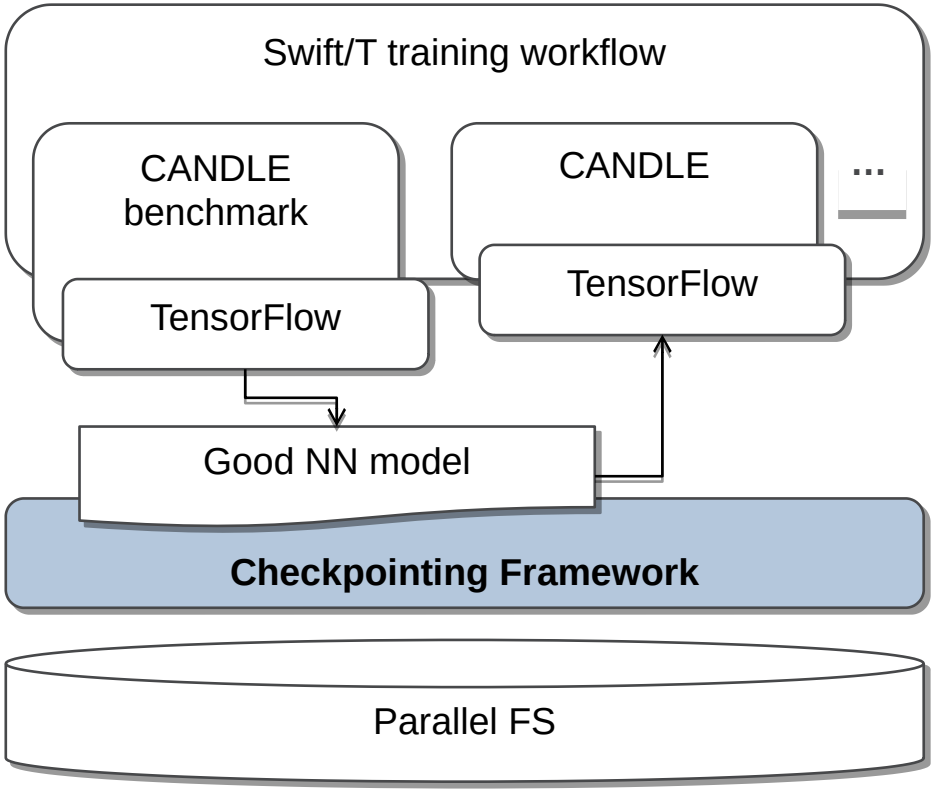
Large-Scale Deep Learning

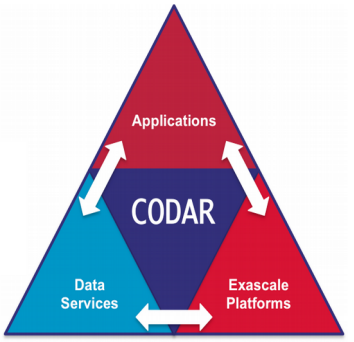


- Key challenge: need to try many possible network configurations, each defined by many possible parameters
- Becomes a big data problem
- Example: CANDLE (more than 650GB/s on Summit pre-exascale machine)

Model Caching for Deep Learning

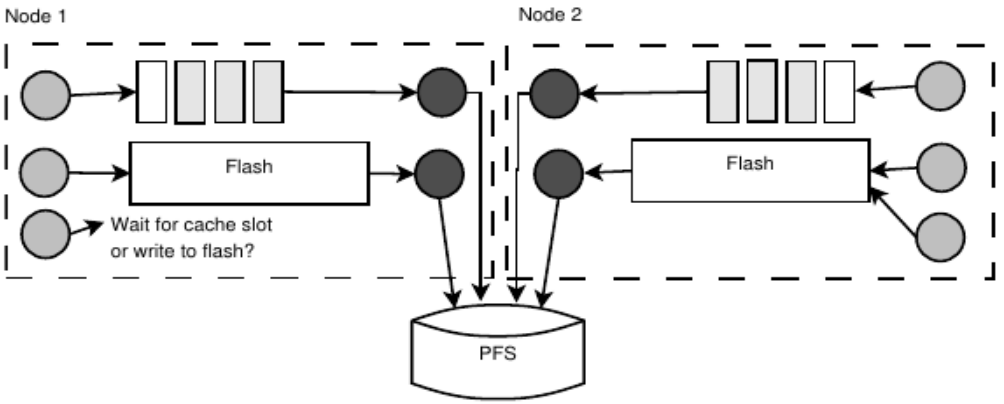
- Deep Learning workflows produce a great number of medium-sized ML models
- Challenges:
  - Cannot keep them on a parallel file system due to I/O bandwidth limitations
  - Need to flush to global FS before end of run, but many models will be discarded
- Solution: Cache these models on compute node storage for *possible* later use
- Idea: Use asynchronous multi-level checkpointing to hide interactions with parallel file system
  - Resist cache failures
  - Flush during runtime to global FS to avoid I/O overhead before end-of-run

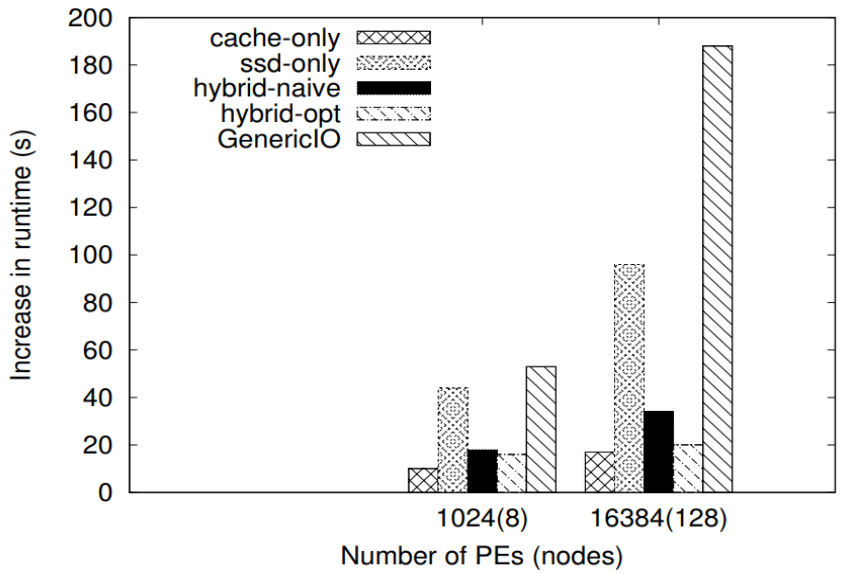




Scaling Deep Learning for Cancer with Advanced Workflow Storage Integration. Justin M. Wozniak, Philip E. Davis, Tong Shu, Jonathan Ozik, Nicholson Collier, Manish Parashar, Ian Foster, Thomas Brettin, and Rick Stevens. Proc. MLHPC @ SC 2018.

Hybrid Local Storage





- Problem: Naive strategies that write to fastest available local storage are suboptimal for producer-consumer patterns
- Example:
  - Nodes equipped with small RAM cache (6 GB) and flash storage (128 GB)
  - Two resilience levels: local and parallel file system (async flush from local)
  - When RAM cache is full, if PFS is faster than flash storage, it is better to wait for RAM cache instead of writing to flash
- Solution: adaptive strategy to pick local storage based on write throughput prediction
- Adaptive strategy can be 2x faster than naive strategy (App: Cosmology)

B. Nicolae et at: VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale
To be presented at IPDPS'19

Thank you!

PS: Looking for interns :)

Contact: bogdan.nicolae@acm.org
Web: http://www.bnicolae.net