IMPROVING COLLECTIVE AGGREGATION FOR HPC AND AI WORKLOADS

Author: Mikaila J. Gossman
Advisor: Dr. Bogdan Nicolae
Advisor: Dr. Jon C. Calhoun
Email: mikailg@g.clemson.edu

MOTIVATION & BACKGROUND

- HPC and AI applications generate and consume massive volumes data—e.g., on 512 nodes, HACC produces 64 TB, VPIC 100 TB, and training a 405B LLM produces ~8 TB per checkpoint
- Limited read/write bandwidth in storage and memory stalls computation, with I/O consuming ~30% of total runtime in some workloads like modern LLMs
- Shared parallel file systems (PFSes) struggle with dynamic, concurrent access patterns, amplifying I/O bottlenecks at scale
- Applications adopt the file-per-process technique to exploit parallel, lock-free I/O operations, but creates challenges at scale due to excessive file creation
- At extreme scale, the proliferation of files can overwhelm metadata servers and creates challenges for users in downstream tasks such as storage, transfer, analysis, and reuse
- Aggregation addresses these issues by consolidating data into fewer shared files
- While aggregation and asynchronous I/O can reduce overhead, their performance is limited by various system- and application-level constraints
- Efficient aggregation is therefore a multi-objective optimization problem, requiring trade-offs across bandwidth, concurrency, access patterns, and storage architecture

KEY CHALLENGES

- Resource Contention: Contention arises when the application competes with asynchronous (background) I/O for local-level resources (CPUs, memory, network links), and across highly concurrent I/O operations (for storage devices), leading to poor performance if not mitigated
- Complex I/O Patterns: Multiple factors influence I/O performance—e.g., access size (large vs. small), number of operations, offset locality, etc. Patterns that diverge from the storage system's preferred access style (e.g., large contiguous writes for PFS) degrade throughput
- Concurrent Interleaving: Under concurrency, even optimal patterns become fragmented or misaligned, amplifying performance issues
- Heterogeneous I/O Storage: Systems often span multiple storage layers (GPU memory, host RAM, node-local SSDs, parallel file systems) each with unique performance characteristics and optimal access patterns. Simultaneous, effective utilization is difficult to achieve
- Number of Concurrent Storage Targets: I/O is often distributed across multiple files, objects, or containers. Using too few targets limits parallelism; too many causes oversubscription and metadata overhead. Thus, tuning for the right level of concurrency is essential

CONTRIBUTIONS

- Analysis of Existing Aggregation Strategies: We evaluate existing aggregation approaches using asynchronous checkpointing as a representative bulk I/O workload, identifying performance bottlenecks under realistic, high-concurrency conditions
- Performance Modeling: We develop a multi-threaded I/O performance model to study how key parameters (e.g., thread count and operation ordering) influence aggregation behavior and guide the design of efficient multi-threaded aggregation algorithms [1]
- Scalable Aggregation for Multi-Level Checkpointing: We propose a novel aggregation strategy based on a two-phase producer-consumer model that balances coordination and throughput, tailored for asynchronous multi-level checkpointing [2]
- Extending Aggregation to LLM Workflows: We explore applying these techniques to large language model (LLM) checkpoint/restore (C/R) workflows, where extreme-scale I/O requires robust, scalable aggregation strategies [Work in progress]

AGGREGATION IN ASYNCHRONOUS CHECKPOINTING

- We first study MPI-IO's implementation of aggregated I/O and naïve POSIX write-at-offset aggregation effects on write throughput performance
- We use VELOC [3], a production-grade HPC asynchronous checkpointing engine, as a representative of a bulk asynchronous I/O workload
- We find that POSIX I/O suffers from false sharing, while MPI-IO's implementation introduces synchronization overheads, making both approaches suboptimal for largescale I/O

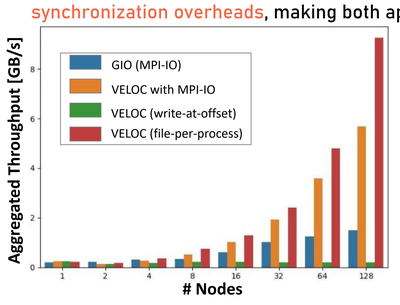

Figure 1: Weak Scalability Analysis of existing I/O aggregation strategies

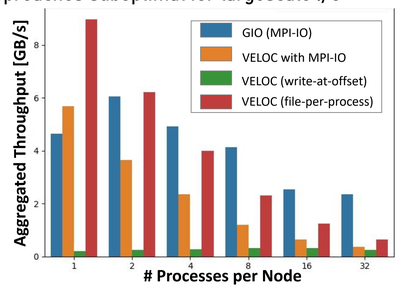

Figure 2: Hard Scalability Analysis of existing I/O aggregation strategies

MODELING PRODUCER-CONSUMER MULTI-THREADED I/O

- We develop OpenMP benchmarks designed to act as a simple, cheap, and convenient proxy enabling a fast evaluation of different combinations of I/O parameters
- We analyze how different parameters (e.g. number of threads, write patterns, and how many threads accessing a file) impact I/O performance in a producer-consumer model
- We show that multi-threaded I/O is necessary to maximize bandwidth utilization and that Interleaved I/O better reduces striping contention, thereby resulting in higher throughput when all threads write to a singular file

	Contiguous	Interleaved
Pros	Easy to implement, no synchronization beyond initial prefix-sum, keeps data organized	Easy to ensure block-aligned writes in the event of uneven files, inherently addresses slow threads
Cons	Susceptible to either load imbalance or competition with other threads if input files are not the same size	Requires synchronization around a write queue, data may get dispersed throughout the file

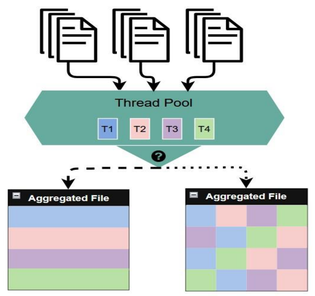

Figure 3: Illustration determining how threads should access the shared file

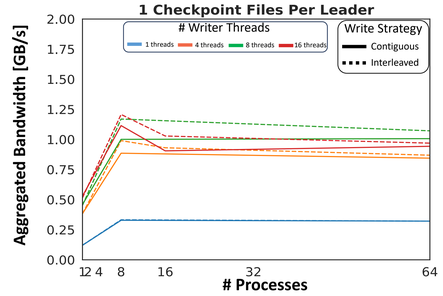

Figure 4: Weak scalability analysis of different write patterns to a shared file (higher is better)

TOWARDS SCALABLE ASYNCHRONOUS AGGREGATION

- We introduce a novel I/O aggregation strategy based on our previous findings
- In microbenchmarks, it achieves up to 2x higher throughput than GIO (an MPI-IO-based checkpointing framework optimized for HACC) and 1.6x higher than ADIOS2
- However, as the number of compute nodes exceeds the number of output files, funneling data through I/O leaders underutilizes PFS resources, leading to performance degradation
- In the HACC application, our strategy delivered 1.2x higher write throughput than GIO, with only 3% checkpoint overhead—a notable reduction from GIO's ~12%

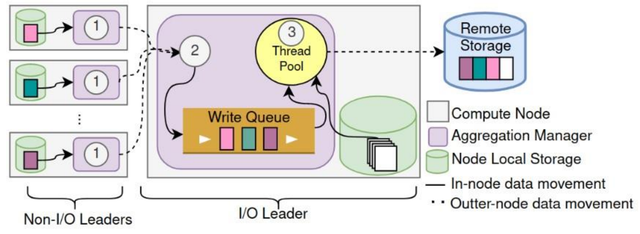

Figure 5: Our streamlined multi-threaded producer-consumer aggregation strategy
(1) represents sender threads, (2) receiver threads, and (3) writer threads

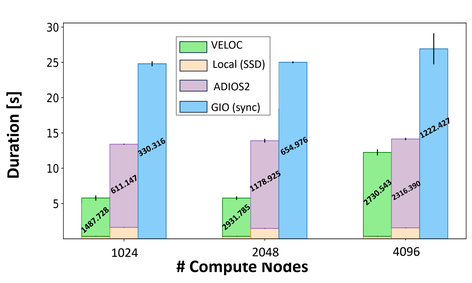

Figure 6: Comparison of our aggregation with leading aggregation strategies in microbenchmark

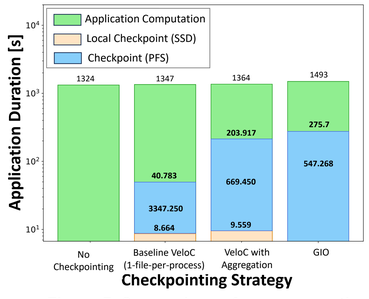

Figure 7: Comparison of our aggregation implementation in HACC application

WORK IN PROGRESS: AGGREGATION FOR LLM C/R

- We evaluate C/R phases of BLOOM 3B using DeepSpeed Megatron as a cheap, but effective way to profile DeepSpeed's native PyTorch-based checkpointing engine
- PyTorch opens/closes files for most writes, creating high metadata overhead
- Restoring a checkpoint incurs ~6x more reads than writing, using smaller buckets
- Seeks dominate restore-phase I/O, degrading performance, especially for small files with low data volume but inefficient disk access

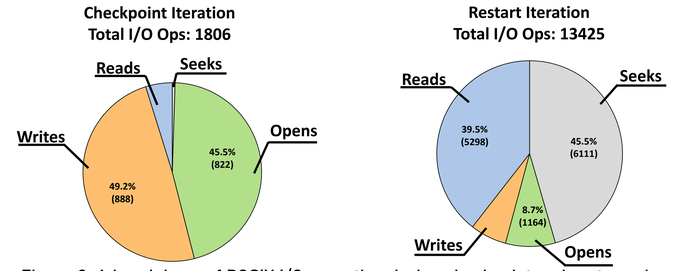

Figure 8: A breakdown of POSIX I/O operation during checkpoint and restore phases of training the BLOOM LLM model using DeepSpeed's PyTorch checkpointing engine

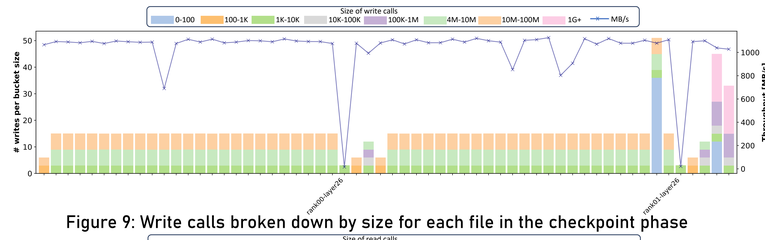

Figure 9: Write calls broken down by size for each file in the checkpoint phase

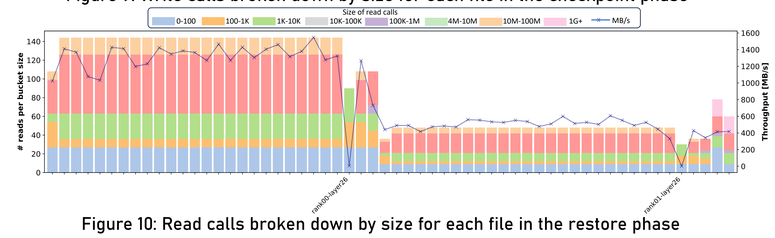

Figure 10: Read calls broken down by size for each file in the restore phase

CONCLUSIONS

- Efficient data aggregation is critical for improving the manageability and performance of I/O on large-scale HPC systems
- Designing robust aggregation strategies requires balancing bandwidth utilization against the risk of overloading PFS components such as metadata servers and storage targets
- We developed a lightweight OpenMP benchmark to help users tune aggregation parameters for their specific systems
- Our producer-consumer-based aggregation model outperformed existing I/O libraries—achieving up to 2x higher write throughput in microbenchmarks and 1.2x in real-world application, HACC
- However, we observed that as compute nodes outscale the number of output files, the strategy underutilizes PFS resources, highlighting the ongoing challenge of balancing throughput and parallelism
- With the rise of AI and the massive data it generates, we will explore how these techniques can be extended to improve the scalability of checkpoint/restart in large language model (LLM) workloads and other data-intensive applications

PUBLICATIONS

[1] Gossman, M., Nicolae, B. and Calhoun, J. 2023. Modeling Multi-Threaded Aggregated I/O for Asynchronous Checkpointing on HPC Systems. *ISPDC'23: The 22nd IEEE International Conference on Parallel and Distributed Computing* (Bucharest, Romania, 2023), 101–105
[2] Gossman, M.J., Nicolae, B. and Calhoun, J.C. 2024. Scalable I/O aggregation for asynchronous multi-level checkpointing. *Future Generation Computer Systems*. 160, (2024), 420–432. DOI:https://doi.org/10.1016/j.future.2024.06.003.
[3] VELOC – https://veloc.readthedocs.io/en/latest/

Argonne National Lab and Clemson University are acknowledged for generous allotment of compute time on the Polaris and Palmetto2 Supercomputers. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114. The material was supported by U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11.357