File Aggregation for Asynchronous Multi-Level Checkpointing

Mikaila J. Gossman
Clemson University
USA
mikailg@g.clemson.edu

Bogdan Nicolae (advisor)
Argonne National Laboratory
USA
bnicolae@anl.gov

Jon C. Calhoun (advisor)
Clemson University
USA
jonccalclemson.edu

ABSTRACT

Checkpointing serves numerous functionalities in modern-day HPC systems and applications. In recent years, synchronous checkpointing that blocks the application until checkpoints are persisted to external storage suffer rising synchronization overheads at scale, resulting in little forward progress by the application. Therefore, asynchronous checkpointing has become more popular by quickly capturing checkpoints locally and flushing them in the background concurrently alongside the application. State-of-the-art solutions like VELOC utilize a file-per-process strategy, which is difficult for users and parallel file systems to manage. We implement a tunable N-to-M aggregation strategy within VELOC, obtaining 2.5x greater throughput than state-of-the-art aggregation library ADIOS2 and 1.5x higher throughput than the naive N-to-1 aggregation currently supported by VELOC.

KEYWORDS

I/O optimization, HPC, asynchronous I/O, checkpoint-restart, file aggregation

1 INTRODUCTION

Checkpointing serves numerous defensive, productive, and administrative purposes, making it a fundamental I/O pattern of high performance computing (HPC) applications [9]. It involves large quantities of processes concurrently persisting critical data to storage. Synchronous checkpointing, like GenericIO [7], blocks the application throughout the duration of the checkpoint, ensuring consistency and eliminating resource competition. However, synchronization overheads on POSIX-compliant file systems (as many are [4]) result in little application progression [8].

Asynchronous checkpointing, like VELOC [8], captures checkpoints locally in a fraction of the time and persists them in the background alongside the application. However, introduces resource contention between checkpointing and the application [10], especially under high levels of concurrency (multi-threading) required to overcome serialization bottlenecks. VELOC adopts an N-to-N (file-per-process) approach, eliminating synchronization overheads, but is difficult for users to manage and suffers metadata bottlenecks in common file system implementations like Lustre [1]. Thus, file aggregation is necessary for asynchronous checkpointing.

One efficient approach to aggregation is N-to-M [2, 7], where a subgroup of processes (M), interact with limited I/O infrastructure on behalf of all processes (N). However, state-of-the-art libraries, like MPI-IO, disregard resource contention and, if made to, suffer frequent synchronization costs of collective operations, thereby unsuitable for asynchronous checkpointing [6]. Other solutions, like ADIOS2 [5], provide adaptable parameters controlling resource consumption, but to our knowledge have not been evaluated in the context of asynchronous checkpointing.

We begin meeting the challenge of devising an optimized N-to-M aggregation strategy specifically for asynchronous checkpointing by:

- Prototyping a adaptable N-to-M aggregation strategy within VELOC
- Evaluating the throughput obtained using our aggregation with the expected peak performance obtained via the system utility call dd and an OpenMP benchmark [3]
- Comparing our aggregation strategy to naive N-to-1 aggregation and state-of-the-art aggregation libraries ADIOS2, and an optimized implementation of MPI-IO realized through the synchronous checkpointing library, GenericIO

2 METHODOLOGY

Our N-to-M implementation allows users to set: (1) the number of desired files (M), (2) number of I/O threads, and (3) size of buffers. Enabling tunable aggregation capable of meeting the varying demands of bounded applications and file systems. In this preliminary work, we evaluate our strategy using when M = # of compute nodes, which isolates performance bottlenecks only associated with interacting with a parallel file system (PFS), thereby eliminating overheads associated with communication across nodes. However, our strategy supports any number $M \leq$ # of compute nodes.

To eliminate idle-time on faster I/O threads, threads collectively process local checkpoint files sequentially in chunk sizes defined by the user. However, this requires synchronization to data structures managing access to local files. Our goal is to characterize how synchronization between I/O threads affect the throughput of an I/O leader.

3 EXPERIMENTAL RESULTS

We use dd and the OpenMP benchmark to characterize the expected maximum aggregated bandwidth (calculated via equation 1)

$$\frac{\text{total checkpoint size [GB]}}{\text{slowest thread [s]}} \tag{1}$$

of a single I/O leader. We set $N = 8$ (a typical number of local processes per node, since compute nodes typically employ 4-8 GPUs) and $M = 1$, since we evaluate our aggregation strategy where each compute node writes 1 aggregated file. We design a custom micro-benchmark simulating a distributed application (more than 1 node) performing a checkpoint under similar conditions, but introduce variance in the sizes (1 GiB ±20%) and set M=8 and N=64, with 8 processes per node.

We set the number of I/O threads equal to the number of local files (8), and set the size of buffers to 64 MB (bounding memory

utilization). We define the same settings for ADIOS2. Our experiments are performed on Oak Ridge National Lab's Frontier, which supports uses AMD 3rd Generation EPYC compute nodes, and a Lustre PFS with a peak aggregated bandwidth (BW) of 14 TB/s.

3.1 Our N-to-M Compared to Expected Peak Performance of N-to-M aggregation with dd and OpenMP

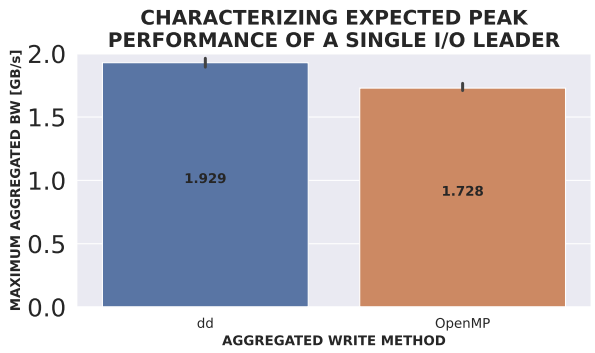

Figure 1: The expected maximum aggregated bandwidth of a single I/O leader under an aggregation scenario where all compute nodes aggregate their own local data to a single file using via dd or OpenMP. (Higher is better)

Figure 1 shows the expected peak BW of N-to-M via dd and OpenMP. OpenMP gets 1.7 GB/s while dd obtains 2 GB/s. Thus, our aggregation should maintain 16 GB/s in our 8 node microbenchmark.

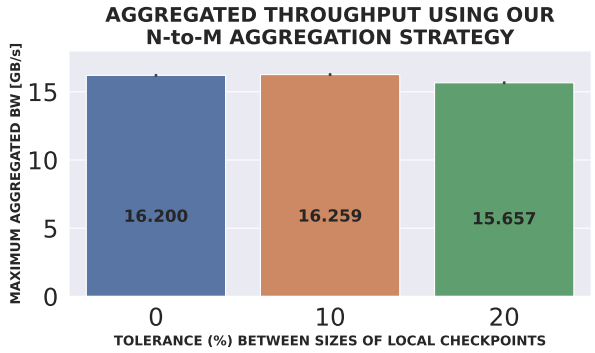

Figure 2: Evaluation of how close our N-to-M aggregation strategy comes to the expected peak throughput (≈16 GB/s) obtained via dd and OpenMP, in an 8 node scenario

Figure 2 plots the maximum aggregated BW of our aggregation implementation when using cumulative fine-grained timers measuring how long each I/O thread spends writing. We obtain more than the expected peak throughput since our implementation mitigates timing imbalance among threads, unlike dd and OpenMP.

3.2 Comparing N-to-M Aggregation Implementations

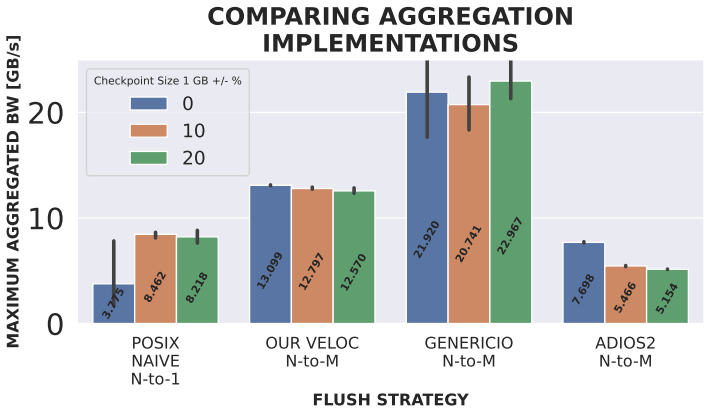

Figure 3: Comparing our N-to-M aggregation strategy against a naive N-to-1 aggregation strategy (currently supported by VELOC) and other, state-of-the-art N-to-M capable aggregation Libraries. Higher is better.

Figure 3 compares our N-to-M aggregation to naive N-to-1 and against other N-to-M implementations via GenericIO and ADIOS2. The throughput is calculated from the application side (which incorporates other function calls besides the writes), resulting in a small drop in throughput compared to results in figure 2. Compared to N-to-1, we get ≈ 1.5× higher throughput and ≈ 2.5× that of ADIOS2. GenericIO gets ≈ 2× higher throughput than ours, however, because they are synchronous, they flush the full 1 GiB region at once, reducing collective calls. This is not feasible in asynchronous checkpointing which needs to limit memory utilization for concurrent workloads.

4 CONCLUSION AND FUTURE WORK

We implement an optimized, tunable aggregation strategy within VELOC capable of maximizing throughput of N-to-M aggregation scenarios, beating some current state-of-the-art N-to-M aggregation libraries, like ADIOS2 when flushing under the same constraints. Even though our collective solution requires synchronization across threads, we find it mitigates timing imbalance, resulting in higher throughput than obtained from optimized system wrappers like dd. Since compute nodes vastly outnumber storage targets in modern HPC systems, we plan to evaluate our aggregation methods when the number of I/O leaders is lower than the number of compute nodes.

ACKNOWLEDGMENTS

Clemson University, Argonne, and Oak Ridge National Lab's are acknowledged for generous allotment of compute time on the Palmetto cluster, Theta, and Frontier Super Computers. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197. The material was supported by U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11.357

REFERENCES

[1] 2003. Lustre : A Scalable , High-Performance File System Cluster.

[2] 2020. *Aggregation.* https://adios2.readthedocs.io/en/latest/advanced/aggregation.html

[3] 2023. *Modeling Multi-Threaded Aggregated I/O for Asynchronous Checkpointing on HPC Systems.* https://github.com/FTHPC/HPC_IO_PROFILING

[4] Michael J. Brim, Adam T. Moody, Seung-Hwan Lim, Ross Miller, Swen Boehm, Cameron Stanavige, Kathryn M. Mohror, and Sarp Oral. 2023. UnifyFS: A User-level Shared File System for Unified Access to Distributed Local Storage. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 290–300. https://doi.org/10.1109/IPDPS54959.2023.00037

[5] Podhorszki Norbert Wang Ruonan Atkins Chuck Eisenhauer Greg Gu Junmin Davis Philip Choi Jong Germaschewski Kai Huck Kevin Huebl Axel Kim Mark Kress James Kurc Tahsin Liu Qing Logan Jeremy Mehta Kshitij Ostrouchov George Parashar Manish Poeschel Franz Pugmire David Suchyta Eric Takahashi Keichi Thompson Nick Tsutsumi Seiji Wan Lipeng Wolf Matthew Wu Kesheng Klasky Scott Godoy, William F. 2020. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *Software X* 12. https://doi.org/10.1016/j.softx.2020.100561

[6] Mikaila J. Gossman, Bogdan Nicolae, Jon C. Calhoun, Franck Cappello, and Melissa C. Smith. 2021. Towards Aggregated Asynchronous Checkpointing. arXiv:2112.02289 [cs.DC]

[7] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, and et al. 2016. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42 (Jan 2016), 49–65. https://doi.org/10.1016/j.newast.2015.06.003

[8] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 911–920. https://doi.org/10.1109/IPDPS.2019.00099

[9] Bogdan Nicolae, Adam Moody, Gregory Kosinovsky, Kathryn Mohror, and Franck Cappello. 2021. VELOC: VEry Low Overhead Checkpointing in the Age of Exascale. *CoRR* abs/2103.02131 (2021). arXiv:2103.02131 https://arxiv.org/abs/2103.02131

[10] Shu-Mei Tseng, Bogdan Nicolae, Franck Cappello, and Aparna Chandramowlishwaran. 2021. Demystifying asynchronous I/O Interference in HPC applications. *The International Journal of High Performance Computing Applications* 35, 4 (2021), 391–412. https://doi.org/10.1177/10943420211016511 arXiv:https://doi.org/10.1177/10943420211016511