Analyzing Performance of File Aggregation in VELOC

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

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

Jon C. Calhoun (advisor),
Melissa Smith (advisor)
Clemson University
USA
{jonccal,smithmc}@clemson.edu

ABSTRACT

As High-Performance Computing (HPC) systems and applications continue to grow in size and complexity, the process of checkpointing to external storage often results in I/O contention and degraded performance. Multi-level asynchronous checkpointing strategies like VELOC (Very Low Overhead Checkpoint Strategy) are gaining popularity among application scientists for the ability to leverage fast node-local storage and flush independently to stable storage in the background. Currently, VELOC adopts a one-file-per-process flush strategy; in large scientific applications, hundreds of thousands of files quickly overwhelm the network bandwidth, filesystem, and application scientists. Thus, applications need to condense the number of checkpoint files while preserving the I/O efficiency of asynchronous multi-level workflows. This work implements different aggregation strategies into VELOC's asynchronous checkpoint runtime and analyzes their impact on performance.

KEYWORDS

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

1 INTRODUCTION

Checkpointing distributed HPC applications is a common task in various scenarios: resilience, job management, reuse of computational states. Typically, checkpoints are persisted to an external repository such as a parallel file system (PFS), whose aggregated I/O bandwidth is limited. Synchronous checkpointing strategies that block applications until all processes checkpoint to a PFS are impractical at exascale [3] due to high overheads. In response, asynchronous multi-level checkpoint systems like VELOC [3] write checkpoints to fast, node-local storage, then flush to the PFS in the background, thereby reducing checkpoint overhead.

While flushing to the PFS, asynchronous checkpointing schemes share resources of the compute nodes (CPU cores, memory, network bandwidth) with application processes. Therefore, an important challenge is to mitigate resource contention to avoid additional overheads. In this regard, VELOC adopts a one-file-per-process asynchronous flush strategy, which has two advantages: (1) it is simple, portable and efficient (no I/O locks); (2) enables an independent strategy for mitigating contention on each compute node.

However, one-file-per-process strategies also have disadvantages, because it generates large numbers of files at scale: (1) some PFS experience metadata bottlenecks when large numbers of files are accessed simultaneously (especially when stored in the same directory); (2) it is difficult for domain experts to manage checkpoints (e.g., move between data centers, verify integrity, use in a producer-consumer workflow, etc.). To alleviate these issues, efforts like GenericIO [1] combine checkpoints from N processes into M files, with M typically set by the application and often fixed to one.

Although effective for synchronous checkpointing [1, 2, 4, 5], such aggregation strategies to our knowledge, remain unexplored for asynchronous checkpointing. The main challenge in this context is designing an efficient aggregation strategy that maintains a high speed of flushing, while also efficiently operating in the background without increasing overheads due to resource contention. This is a non-trivial challenge that involves a co-design of both aggregation and mitigation strategies. In this paper, we take a first step towards solving the aforementioned challenge and summarize our contributions as follows:

- Implement two aggregation strategies on top of VELOC that combine all local files into a single, shared file on the PFS.
- Evaluate our strategies against VELOC's original file-per-process asynchronous strategy and GenericIO's optimized synchronous aggregation strategy. We use these results to characterize aggregation in asynchronous multi-level workflows and discuss ideas for designing a designated strategy for aggregation specifcally for asynchronous multi-lvevel checkpointing.

2 APPROACH

Our goal is to design an efficient aggregation strategy that maintains high write throughput of VELOC's default implementation, and mitigate interference by reducing competition for resources between the application and background I/O processes. To this end, we implement two naive aggregation strategies in VELOC: one based on POSIX (Section 2.1) and another using MPI-IO (Section 2.2). Using these strategies as a baseline, we characterize how aggregation impacts the checkpointing process and use this to devise more sophisticated methods of aggregation.

2.1 POSIX

The first strategy we implement is a POSIX-based aggregation strategy that aims to highlight the bottlenecks during asynchronous flushing. This strategy leverages the *multi-level* checkpointing

ACM Student Research Competition '21, November 16–18, 2021, St. Louis, MO

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

support introduced by VELOC [4]: (1) each process writes its checkpoint in a blocking fashion to a file on node-local storage (denoted the *local phase*); (2) then, the process continues with application computation and a separate asynchronous post-processing engine (denoted the *active backend*) running on each compute node writes the node-local files to an offset in a shared file located on the PFS (denoted the *flush phase*). The offset of each local checkpoint in the remote file is calculated via parallel prefix-sum algorithm over all participating processes. Note that with this strategy, the active backend running on each node can parallelize writing the local checkpoints using a number of I/O threads that can be used to match the desired trade-off between application slowdown and flushing duration can be configured for each active backend.

2.2 MPI-IO

Aggregation incurs significant overhead due to *false sharing*: to facilitate parallel processing, files are striped across different I/O servers. If writes are not stripe-aligned, processes will compete for access to the I/O servers. To alleviate false sharing, MPI-IO collective calls allow processes on a communicator to collaborate and rearrange requests such that writes to the shared file are stripe-aligned, thereby reducing overhead. However, since this is a collective operation, it assumes all processes are participating in the checkpoint, thus all data needs to be ready *simultaneously*, introducing synchronization. To avoid this we perform a collective write across all the active backends. However, each active backend may contribute multiple files from the processes co-located on the same node. Thus, we propose a multi-phase solution that invokes multiple successive MPI-IO collective write calls, one for each node-local checkpoint.

3 PRELIMINARY RESULTS

We develop a benchmark that spawns a varied number of processes, each writing 1 GiB checkpoint files. Results are obtained on Argonne National Lab's Theta, a Cray XC40 11.60 petaflops system, equipped with a Lustre PFS [5, 6]. The results show the throughput for the two aggregation implementations, as well as two other approaches: VELOC's [3] default file-per-process, and GenericIO's (GIO [1]) synchronous aggregation strategy. The local phase is the time it takes all VELOC methods to checkpoint to local storage, while GIO writes directly to the PFS in all cases. The flush phase is the time it takes all methods to write checkpoints to the PFS. All aggregation methods aggregate files to $N \rightarrow 1$.

Figures 1 and 2 show local and total throughput respectively for the aforementioned strategies when scaling the number of nodes and processes per node set to 1. These results show the prefix-sum operation introduces negligible overhead during the local phase, thus we can conclude aggregation is unlikely to introduce latency during this phase. During the flush phase, the POSIX-based method suffers from false sharing. The MPI-IO method maintains high throughput as MPI-IO condenses and coordinates writes to the PFS.

Figures 3 and 4 show the bandwidth and flush time when using 128 total processes; varying the number of processes per node. From these results, we see that the local phase maintains high throughput in our aggregation method. During the flush phase, performance of POSIX implementation remains mostly unchanged; although, there is slight improvement as more processes are added

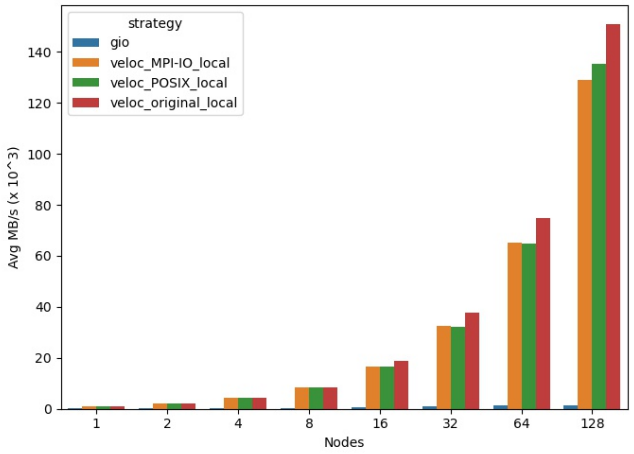

Figure 1: Throughput for local checkpoint phase when scaling compute nodes and processes per node set to 1. Note GIO is writing directly to the PFS. Higher is better

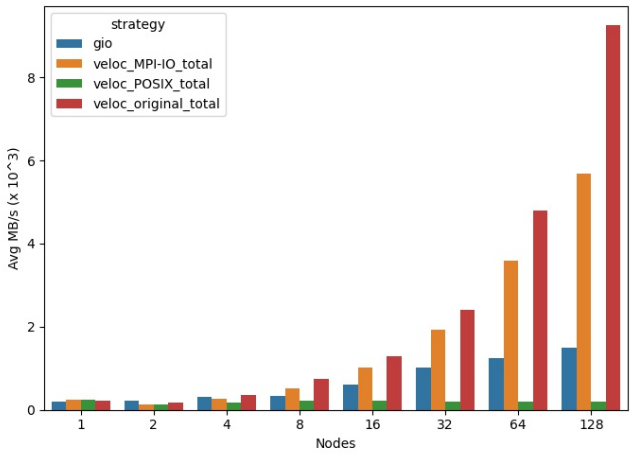

Figure 2: Throughput during flush phase (async) when scaling compute nodes and processes per node set to 1. GIO is blocking and overlaps with counterpart from Figure 1

per node, spreading out write requests to the PFS. For our MPI-IO implementation, serialization starts to dominate throughput as processes per node increase.

4 CONCLUSION

We implement two asynchronous aggregation strategies and compare benchmark performance to VELOC's asynchronous one-file-per-process strategy, and GenericIO's synchronous aggregated strategy. Our results show that a simple aggregation strategy based on POSIX or MPI-IO is not sufficient and there is significant room for improvement to reach and potentially surpass the default, embarrassingly parallel one-file per MPI rank flush strategy.

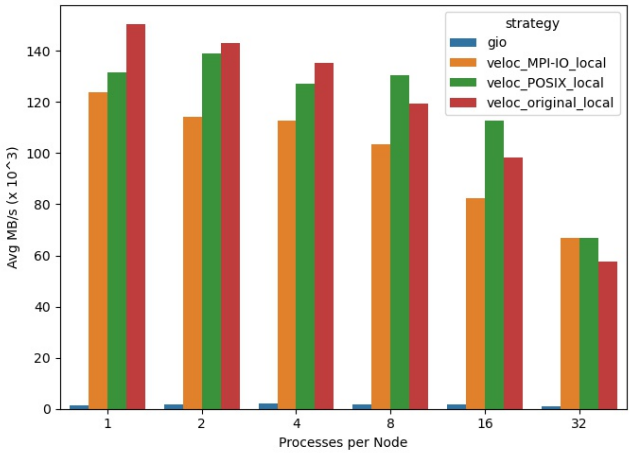

Figure 3: Throughput for local checkpoint phase for an increasing number of processes per compute node. Note GIO is writing directly to the PFS. Higher is better

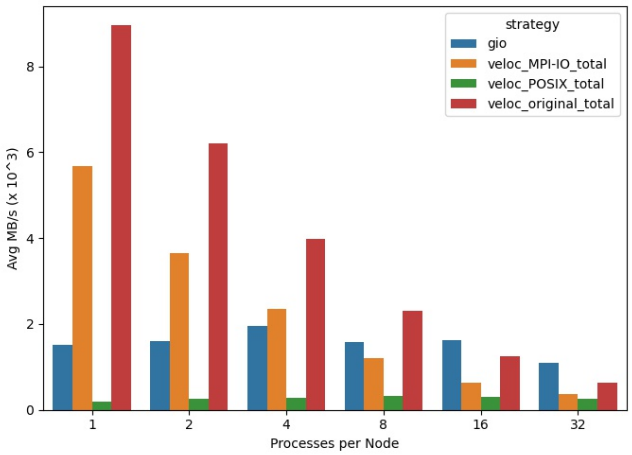

Figure 4: Throughput during flush phase (async) for increasing number of processes per compute node. GIO is blocking and overlaps with counterpart from Figure 3

ACKNOWLEDGMENTS

Clemson University is acknowledged for generous allotment of compute time on the Palmetto cluster. 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] 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

[2] Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. de Supinski, and Rudolf Eigenmann. 2012. MCREngine: A scalable checkpointing system using data-aware aggregation and compression. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1109/SC.2012.77

[3] 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

[4] 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

[5] 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

[6] Rick Zamora. [n.d.]. Optimizing I/O at ALCF: Performance and Best Practices.