A Weak Scalability Study of File Aggregation in Asynchronous, Multi-Level Checkpointing

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

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

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

1 INTRODUCTION

Checkpointing is a common pattern of many scientific HPC applications that facilitates use cases such as: checkpoint-restart based fault-tolerance, enabling job preemption, verification and exchange of intermediate results, etc. It is also a difficult pattern that involves multiple distributed processes writing large amounts of data concurrently, which puts pressure on the I/O subsystems. Checkpointing represents up to 80% of the I/O load of HPC systems [4] and this trend is predicted to continue. Therefore, the performance and scalability of checkpointing is critical to HPC systems.

The most widely used checkpoint strategies are synchronous: these strategies block the application until checkpointing is complete to stable storage. However, this leads to unacceptable overheads, which not only increase the runtime of HPC applications, but may also cause undesired interruptions that delay other tasks (e.g., analytics or training of AI models based on intermediate results). To reduce the checkpointing overheads, asynchronous checkpointing strategies [3] write to fast, node-local storage in a blocking fashion (which causes negligible interruptions), then flush the checkpoints to persistent storage in the background as the application resumes.

To maximize the potential of leveraging concurrent I/O operations, asynchronous checkpointing runtimes like VELOC [3] adopt a file-per-process write strategy. This allows a simple I/O management strategy in the background that is both fast and minimizes interference with the application due to competition for resources. However, this also leads to a large number of files, which may become difficult to manage from user perspective and/or introduce metadata bottlenecks (especially on parallel file systems). Therefore, it is important to aggregate checkpoints into a small number of files/objects that are easier to manage and/or are optimized for the layout of data repositories.

The problem of checkpoint aggregation has been extensively studied for synchronous checkpointing but remains virtually unexplored in the context of asynchronous checkpointing, where it is non-trivial due to resource contention [5] during background I/O operations.

This poster focuses on how to alleviate this issue. It discusses two asynchronous checkpoint aggregation strategies and presents a series of preliminary results that evaluate their effectiveness using synthetic benchmarks.

2 POSIX AGGREGATION

The POSIX method is an extension of VELOC's original implementation. We calculate the global offset in the shared file via parallel prefix sum during the *local phase* of checkpointing. Application processes return to computation, and separate processes spawned via *the active backend* asynchronously flush the local checkpoints to the calculated offsets with POSIX writes.

We tested our aggregation strategies on a custom microbenchmark; it spawns N ($1 \rightarrow 4096$) processes that write 1 gigabyte (GB) checkpoint files to be persisted to the PFS. Experiments ran on Argonne National Lab's Theta System [1]. The POSIX method suffers from *false sharing*, where only one process can access a *stripe* (storage disk) of a shared file at a time. When writes are not stripe-aligned, this introduces I/O bottlenecks.

3 MPI-IO AGGREGATION

MPI-IO is used by optimized synchronous aggregation libraries, like GenericIO [2]. MPI-IO utilizes collective calls, which boost performance by aggregating and coordinating I/O requests from processes on a communicator, eliminating false sharing and reducing the number of processes interacting with the PFS.

Collective operations require synchronization (for data exchange) and single calls do not allow writes to discontiguous regions of a file. But each backend may contribute multiple files at discontiguous offsets. Our experiments show a trade-off between the cost of synchronization and serialization (to reduce synchronization), and that it was more beneficial to reduce the synchronization rather than serial cost. Still, serialization leads to performance bottlenecks.

4 CONCLUSION

Our results found that simple POSIX and MPI-IO strategies are not suitable for aggregation in asynchronous checkpointing. POSIX suffers from false sharing and MPI-IO suffers from serialization and synchronization. Currently, we are working to devise a novel aggregation strategy that addresses false sharing, while eliminating some of the required synchronization for using MPI-IO.

REFERENCES

[1] Top500 2021. *Theta - Cray XC Cray XC40, Intel Xeon Phi 7230 64C 1.3GHz, Aries interconnect.* Top500. https://www.top500.org/system/178926/

[2] 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

[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] Fabrizio Petrini and Wu Feng. 2000. Scheduling with global information in distributed systems. 225–232. https://doi.org/10.1109/ICDCS.2000.840933

[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.