

## Exploring Solutions to ASC Simulation Code Input/Output Issues

Steve Hodson, James Nunez, HPC-5

**T**here is a conflict between the Los Alamos National Laboratory Advanced Simulation and Computing (ASC) simulation codes and the supercomputers on which they run. The codes can run anywhere from weeks to months to years, yet the computer systems rarely stay up for this amount of time. One method for the simulations running to make progress is for the codes to periodically write files to storage that have all the information in them to restart the run. The stored files are called restart, or checkpoint, files. Normally, each process writes all characteristics for all elements for which it is responsible to a single file. Thus, if a machine goes down, the code can restart close to the place where it stopped with the use of the information in the checkpoint file. In this manner, days, weeks, or months of work are not lost. Since the checkpoint files can be very large, on the order of gigabytes to terabytes, the time to write a checkpoint file is anywhere from seconds to tens of minutes, and the user must balance this cost against the mean time to interrupt (MTTI) of the machine to figure out how often to write a checkpoint file. In addition, the MTTI will vary by machine, maturity of machine, by the size of the job, and the checkpoint time will depend on the application.

Unfortunately, the manner in which many codes write checkpoint files is with very small writes, on the order of tens to a few hundred kilobytes, and the writes are normally not aligned on page boundaries. Not only do they write in this small and unaligned manner, but all processes write to one portion of a single file at one time, i.e., the writes are strided. The codes have a variety of good reasons to write in this manner; it being the simplest way to restart a job from N processors to M processors, and a friendly format for visualization.

Since the simulations cannot afford to lose data, most file systems employ a reliability scheme called RAID 5 that can rebuild data in the event of a failed disk drive by keeping the parity of data stored across all drives. Due to this reliability and the code's I/O (input/output) patterns, every write to disk is possibly a fetch-on-write or read the blocks on disk, update these blocks with the new data, and write to disk and thus is more time consuming. For this reason the HPC-5 File System team explored the benefits and risks of a nonparity-based reliability scheme.

With real application in hand, the file system team, including the authors, tested a variety of I/O techniques to speed up the write time for real applications. Steve Hodson brought a deep understanding of the underlying physics of the applications to the team, and James Nunez brought an understanding of I/O middleware and file systems. The first step in eroding the decade-long problem of slow write bandwidth due to small unaligned I/O, was to analyze the I/O patterns of the application. Analyzing the application's I/O patterns entailed instrumenting the I/O routines and coming up with a profile for a typical run. With that analysis, potential areas that would impact the application's time for writing checkpoint files were found. The initial problem the authors were given was a code that was getting 1 MB/s of write bandwidth, which is intolerably slow. The analysis showed that all processes were writing to a small region of a single file with very small blocks of data. The team did a study of the impact of data aggregation at the middleware (MPI-IO) level; i.e., sending data to a small number of processes and having only that subset write to the file. This study confirmed previous results that having many writers can actually slow I/O

rates. Reducing the number of writers led to a 20-time improvement in bandwidth; to 20 MB/s. Due to rigorous testing and parameter studies, the data aggregation parameters were made the default in Los Alamos Message Passing Interface (LA-MPI) on all production machines.

Even this improved data rate was unacceptable. So Steve continued to search for ways to improve the application's I/O rates. In conjunction with testing James was doing, they realized that their bandwidth results for very similar tests were not matching. An analysis of the differences in their testing showed that they were using different versions of MPI, MPICH vs LA-MPI, which led to the discovery that LA-MPI had a very inefficient implementation of a subroutine that was called hundreds of times by the I/O routines. Once the LA-MPI team optimized the I/O subroutine, the application's write rate increased to approximately 50 MB/s.

Next, Steve undertook the job of activating asynchronous I/O in MPI and a study of how this would affect the application's I/O rates. Although the work with asynchronous I/O helped the bandwidth, it was clear that there were other factors impeding the I/O performance.

With help from the file system vendor Panasas, the team tested a nonparity-based file layout called RAID0. The team conducted a study of collective, independent synchronous, and independent asynchronous I/O against the alternate file layouts. The study concluded that the RAID0 layout alone helps the application, but combined with

asynchronous independent I/O (Fig. 1), the application can achieve over 450 MB/s!

The next step for this team is to work with Panasas to bring a nonparity-based reliable file layout into its product and into the Laboratory. This work has already begun, and testing on a RAID 10, mirror and stripe data across multiple disks scheme, already looks promising.

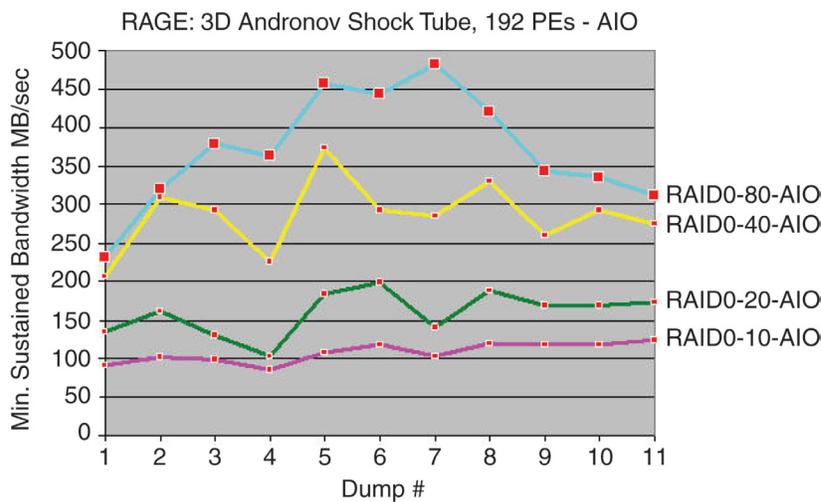
**Conclusion**

Through persistence, experimentation, and methodical testing, significant progress has been made toward understanding and solving the small and unaligned I/O problem that plagues many of ASC major codes. The File System team was able to demonstrate an average of 350 times improvement in write bandwidth for a major code team.

*For more information contact Stephen Hodson at [swh@lanl.gov](mailto:swh@lanl.gov).*

**Funding Acknowledgements**

This research was supported by the NNSA tri-Lab Advanced Simulation and Computing Program.



**Fig. 1.** The top turquoise line shows the dramatic improvement in speed of the application write time on a super-computer when RAID0 is used in combination with a specific I/O method.