I/O Characteristics of Scientific Applications

Chen Wang[1] chenw5@illinois.edu
Adam Moody[2] moody20@llnl.gov
Elsa Gonsiorowski[2] gonsie@llnl.gov
Kathryn Mohror[2] kathryn@llnl.gov
Marc Snir[1] snir@illinois.edu

I. Motivation

Our goal is to analyze and understand the I/O characteristics of HPC applications in order to to identify opportunities for improving application I/O performance. For example, can we relax POSIX semantics to make I/O faster and still meet an application's requirements? What functions or attributes (e.g., access time, modification time) do applications actually use and need? Can we omit support for some functions and attributes in favor of performance when building a file system?

II. Method

In this preliminary study, we generated I/O traces from 25 real-world, scientific applications with Recorder [1], a multilevel tracing tool that generates detailed I/O information from application executions. We extended Recorder in three ways: we added support for a wider range of I/O function calls, we designed and implemented a compressed encoding schema that significantly reduces log file sizes by exploiting the redundancy of intra-process I/O operations, and we developed a visualization tool that generates a detailed I/O report for an application. For example, Figure 1 shows the file offsets accessed by each rank for the Flash application.

We highlight some of our early observations here:

- Application access patterns are mostly read-only or write-only (first column in Table I). Based on this, strict POSIX consistency may not be necessary. In parallel file systems, file accesses are normally protected with global locks that can cause performance penalties. The question that remains is: to what extent can we relax the POSIX semantics without breaking the applications?
- Most files are never read back once written or closed. In Table I, R→R, R→W, W→W and W→R indicate read-after-read, write-after-read, write-after-write and read-after-write on the same file offsets. Our results show that applications read the same file offsets repeatedly. However, few or no applications ever perform writes after reads or reads after writes, which suggest we can discard the data in file system buffers once written.
- Most applications have contiguous read/write patterns (Figure 2). Applications can benefit from read-ahead caches but need to set the cache block size accordingly.
- Many attributes, e.g., last access time, last modification time), and last change time, are never used by applications directly. However, faithfully implementing them imposes non-trivial overheads due to frequent metadata operations. Thus, it might be reasonable to ignore some of those attributes when building a file system.

Currently all applications in our study were run on a small scale with default configurations. In our future work, we will test with more configurations, e.g., with different problem sizes and MPI hints, and at larger scales. We still have many questions to answer to complete our study. For example, what is the minimum POSIX semantics requirement for applications? How much do reads skip around files? What is the read cache/paging effectiveness for a given cache/page size?

App	R/W only	R→R	W→W	R→W	W→R
Flash	✓	×	×	×	×
Nek5000	✓	S;M	×	×	×
LAMMPS	✓	×	×	×	×
VASP	✓	S;M	S	×	×
QMCPack	✓	M	S;M	×	×
ENZO	×	×	×	S	×
LBANN	✓	M	×	×	×

TABLE I. I/O Patterns

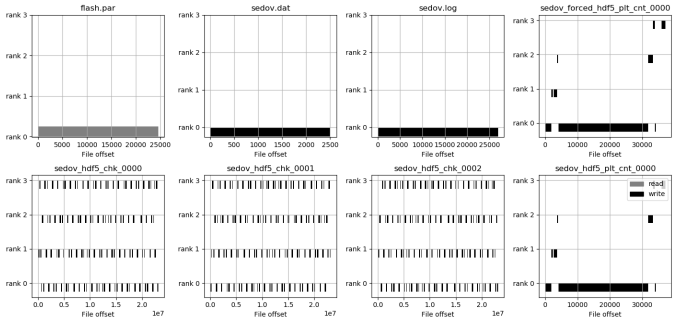

Fig. 1. Flash - Offset vs Rank

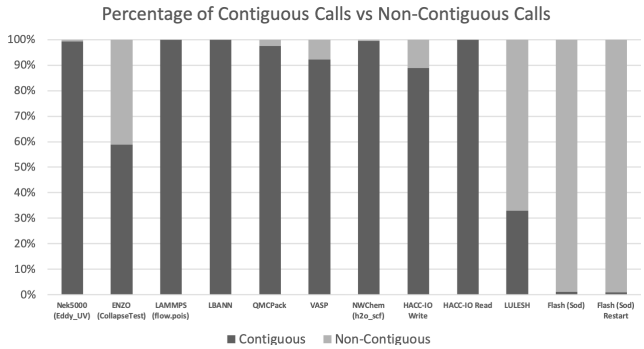

Fig. 2. Percentage of contiguous I/O operations in different applications

References

[1] Recorder. [Online]. Available: https://github.com/uiuc-hpc/Recorder

[1]University of Illinois at Urbana-Champaign
[2]Lawrence Livermore National Laboratory