A Classification of Parallel I/O Toward Demystifying HPC I/O Best Practices

Robert Sisneros
National Center for Supercomputing Applications
Uinversity of Illinois at Urbana-Champaign
Urbana, Illinois, USA
Email: sisneros@illinois.edu

***Abstract*—The process of optimizing parallel I/O can quite easily become daunting. By the nature of its implementation there are many highly sensitive, tunable parameters and a subtle change to any of these may have drastic or even completely counterintuitive results. There are many factors affecting performance: complex hardware configurations, significant yet unpredictable system loads, and system level implementations that perform tasks in unexpected ways. A final compounding issue is that an optimization is very likely specific to only a single application. The state of the art then is usually a fuzzy mixture of expertise and trial-and-error testing. In this work we introduce a characterization of application I/O based on a combination of job-level and filesystem-level aggregation. We will show how this characterization may be used to analyze parallel I/O performance to not only validate I/O best practices but also communicate benefits in a user centric way.**

***Keywords*-Parallel I/O; Filesystems; HPC**

I. Introduction

The process of optimizing parallel I/O can quite easily become daunting. By the nature of its implementation there are many highly sensitive, tunable parameters and a subtle change to any of these may have drastic or even completely counterintuitive results. Additionally there are many other factors affecting performance: complex hardware configurations, significant yet chaotic and unpredictable system loads, and system level implementations that carry out tasks in unexpected ways. A final compounding issue is that an optimization is very likely specific to only a single application. The state of the art then is usually a fuzzy mixture of expertise and trial-and-error testing. While this is sufficient to create documented best practices, the aforementioned fuzziness bleeds over in unfortunate ways; we have no reliable system for classifying application I/O that easily maps applications to best practices. In this paper we outline the steps we have taken to demystify parallel I/O best practices as they apply to our system: the large scale Lustre filesystem of the Blue Waters supercomputer [1].

We believe the above issues typically lead to a fractured set of best practices composed of advice for users that is either rooted in experience and regards how to improve performance (e.g. how to tune Lustre parameters), or is general and really representative of how to best utilize resources with regard to all users on the system (e.g. benefits of shared file parallel I/O). In reality, variability among applications regarding aggregation, number of variables written, storage patterns both on disk as well as in data structures, domain decompositions, etc. typically means that optimizing I/O requires the hand tuning of an I/O expert. This hand tuning focuses on avoiding the universal pitfalls for achieving good performance: conducting I/O that is not stripe aligned and OST (object storage target, i.e. disk) contention among resources performing parallel I/O. These considerations serve as the motivators for this work. First, we believe our efforts as creators of best practices are too focused on aspects that we are unable to usefully generalize and too dependent on the knowledge of intricacies of parallel I/O. We therefore ignore attempting to provide advice on how to optimize application performance and instead generalize all we can at a higher level.

The major contributions of this work are as follows. We have

- created a framework for abstracting the I/O models of an application,
- developed a new benchmarking code for iterating and testing this set of I/O models, and
- devised a system of visually representing the benchmarking results that provides a single context within which best practices are validated.

Critically, these fit together in such a way as to elucidate the relationship between I/O models including those commonly addressed in best practices: file per process and single shared file.

In the remainder of the paper we will provide a brief overview of typical best practices and the general parallel I/O resources use in the creation thereof in Section II. We will discuss particulars of benchmarking I/O models and visualizing the results in Sections III and IV, respectively. In Section V we will outline representative tests conducted and discuss results and will conclude in Section VI.

II. Background

In this Section, we present representative examples of relevant (to our center) HPC parallel I/O best practices. We also provide typical resources referenced and utilized in their creation.

Our intention is to apply knowledge gained from this work in the managing of the Blue Waters supercomputer.

Blue Waters is a Cray XE6-XK7 supercomputing system. The system has 26 PB online disk capacity and two types of compute nodes: XE6 and XK7. There are 22,640 XE6 nodes and 4,224 XK7 nodes connected via Cray Gemini interconnect. The tests for this paper were however run on the Blue Waters test and development system, JYC. JYC contains 54 XE6 nodes and 28 XK7 and has eight available OSTs across which we may stripe files.

For excellent examples of public facing parallel I/O best practices writeups, we direct the reader to the pages from the National Institute for Computational Sciences (NICS) [2] and the National Energy Research Scientific Computing Center (NERSC) [3]. A comparison of these sites illustrates commonalities of utilizing large scale Lustre file systems effectively. These sites focus on introducing parallel I/O at a high level and discussing how to configure Lustre stripe settings. Contrasting these sites provides insight into how emphasis may easily very among HPC centers. While NERSC's page dives into tuning MPI-IO [4], NICS's page provides extensive benchmarking results to demonstrating various benefits of stripe settings and I/O patterns. Discussed or not, we believe such benchmarking results are central to the understanding of parallel I/O and the creation of general best practices.

The typical benchmark for Lustre filesystems is the Interleaved or Random (IOR) I/O Benchmark [5]. IOR was developed at Lawrence Livermore National Laboratory for measuring read/write performance of parallel file systems. There are a number high-level customizable parameters that may be set; these include file size, I/O transaction size, sequential vs. random access, and single shared file vs. file per process. There are also several flags for fine tuning a particular run that allow a user to have IOR perform tasks such as consume system memory (in addition to measured I/O tasks) or keep temporary files created and written for tests. In addition, the diverse repertoire of configurations may be applied to any of IORs supported APIs: POSIX, MPIIO, HDF5, and NCMPI. These features make IOR an ideal benchmark for an HPC facility and it is indeed widely used in that community [6], [7], [8]. However, as we will detail in Section III, this benchmark is inadequate for our purposes. While there are other benchmarks, to our knowledge none are as widely used as IOR nor offer functionalities suitable for our needs.

III. Comprehensive I/O Model Benchmarking

In our experience, I/O benchmarking at HPC centers may be categorized as either a general benchmark such as IOR testing only two I/O models, file per process and single shared file, or an application specific I/O kernel. The latter is clearly not generally viable and will warrant no further discussion in this paper. We believe the simple fact that there are many possible I/O models between file per process and single shared file makes the former inadequate in providing accessible understanding of parallel I/O. Furthermore, with each test corresponding to a single execution the norm for benchmarking is hold constant block/transfer sizes across varying processor counts or to hold constant processor counts across varying block/transfer sizes. In either of these cases, Lustre stripe settings are configured appropriately, or even varied as well to collect additional data. From one test to the next resulting data almost always represents a different amount of actual I/O performed and in the case of also varying Lustre settings resulting data is difficult to display intuitively.

We believe the key to understanding parallel I/O is in benchmarking according to the following two criteria:

- I/O models between file per process and single shared file must be measured, only then may the actual relationship between an I/O model and performance be understood.
- Such benchmarking provides comparable context only when the overall size of I/O being performed is constant across runs.

Our model for an application is therefore quite simple: an amount of I/O to be performed. Given such an amount along with a maximum number of nodes and processors per node available to perform the I/O we want to iterate over all possible I/O models within reason.

We now discuss the answer to "Which I/O models are within reason?". We have settled on a space of possible I/O models bounded by the following three: serial I/O, file per process I/O, and single shared file I/O. The space spanned by these additionally include many combinations of f files per node which may be shared across m nodes. For example, we certainly want to cover file per node I/O (which is often mentioned as an ideal target for aggregation in I/O best practices) but also other non traditional I/O patterns such as file per two nodes. A more complex case: for eight processors per node, there are four files per node and files are shared across four nodes. Therefore, there are 8 processors writing to each file (two from each of the four processors writing to the file). We think of the differentiation among tests to relate to aggregation, serial I/O is complete aggregation within a job and on the filesystem, while single shared file is no aggregation within a job but total aggregation on the filesystem. File per process employs no aggregation.

There are clearly many potential I/O models; in each of the tests described in Section V there are a total of 315. While our initial intention was to utilize existing benchmarking codes to simulate these non traditional I/O models, examples such as the above "more complex case" made this impossible without being able to coordinate simultaneous execution of multiple benchmarks both across and within nodes which is not typically supported when launching jobs on HPC resources and certainly not on Blue Waters. We therefore created a custom benchmarking code that

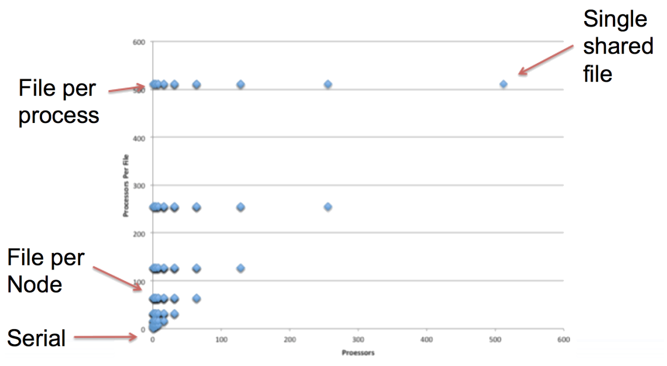


Figure 1: Direct display of 315 I/O models in the space of the number of processors performing I/O vs. the number of processors writing to each file.

takes as input only the three aforementioned parameters: I/O size, maximum number of processors, and maximum number of processors available per node. Our code then iterates over the various I/O models, always writing the same I/O size, and measures POSIX write throughput with parallel writes synchronized by hand with MPI barriers. Furthermore our code need only be executed once without iterations necessary through scripting. With regard to Lustre settings, we simply ensure stripe alignment with the largest possible stripe size via `lustreapi`, the C language API for configuring Lustre stripe settings.

Upon analyzing results of a test run we selected the following two axes to classify a specific I/O model: the number processors (or units) participating in the I/O vs. the number of processors writing to each file. Each of these have a direct relationship to two types of aggregation discussed above. Figure 1 shows a scatterplot of 315 I/O models displayed in this space with common I/O models highlighted therein.

IV. Visualizing Benchmark Data

Figure 1 leaves much to be desired as a visual representation of a parallel I/O classification. In this section we describe and the few simple steps for improving this representation to enable direct performance comparisons among multiple I/O models. Figure 2 is an example of the improved representation.

First, the we take the base two logarithm of each axis to move successive points next to each other visually. This works as each contributing processor must write an amount of I/O that evenly divides the full I/O size. We then treat the resulting combinations of (log) processor counts as the bins of a 2D histogram. From there, we may select any number of reduction operators to condense each bin to a single value and color accordingly. In Figure 2 the typical count is used, this image is colored by the count of the number of I/O

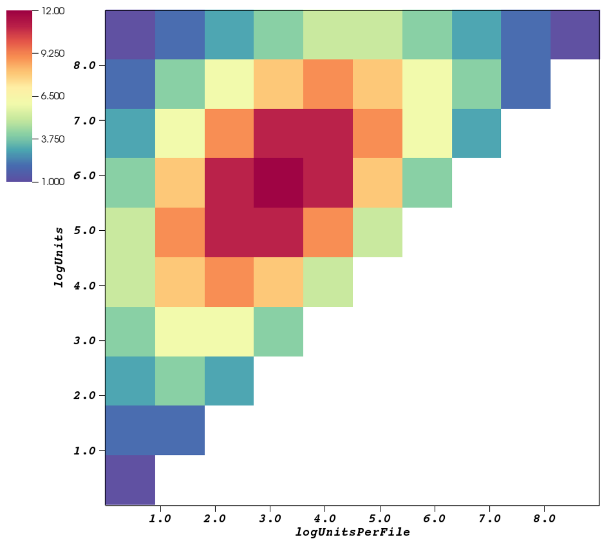


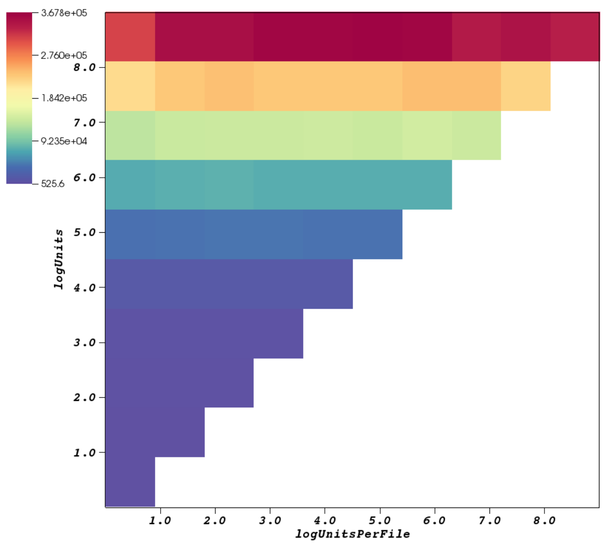
Figure 2: The improved representation, a 2D histogram over the base two logarithm of number of processors performing I/O vs. the logarithm of the number of processors writing to each file. In this image, each bin is colored according to the count of the number of I/O models the bin contains. For 315 tests, the maximum number of models in any bin is 12.

models, or 315 total, falling in each bin. In this example, the maximum number of models for any bin is 12.

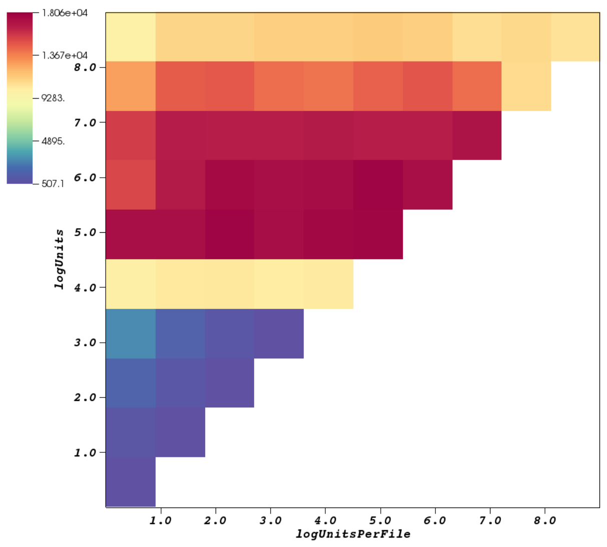
There are several benefits of this classification. When viewed as a 2D space, any job fits nicely into this context. Also, these axes do not form an orthogonal basis and collapses in an interesting way; this allows for the clarification of the relationship among I/O models, particularly those commonly used. Furthermore, looking at I/O in this way makes it necessary to test uncommon/unused I/O models, say file per half job. We believe extensive testing in this framework will help do two things: provide the "missing links to be able to conceptualize application I/O globally, and provide a natural framework for highlighting machine-specific capabilities such as available number of OSTs. We explore this in the next Section.

V. Results

In this Section we detail the two experiments conducted on the Blue Waters test and development system, JYC. We used JYC as we could easily dedicate the machine to our tests thereby providing the proof of concept of our classification. For each test, a maximum number of 32 nodes (of the available 54 XE6 nodes) and a maximum number of 16 cores per node (available floating point cores) are used. The difference of the two tests are the I/O sizes:



(a) 32MB I/O throughput.



(b) 32MB I/O efficiency.

Figure 3: 2D histograms of writing 32MB on various node/core counts from 1 to 512 cores across 32 nodes. Each bin is a maximum of throughput (a) or throughput per node hour (b) and represents an I/O model or set of models.

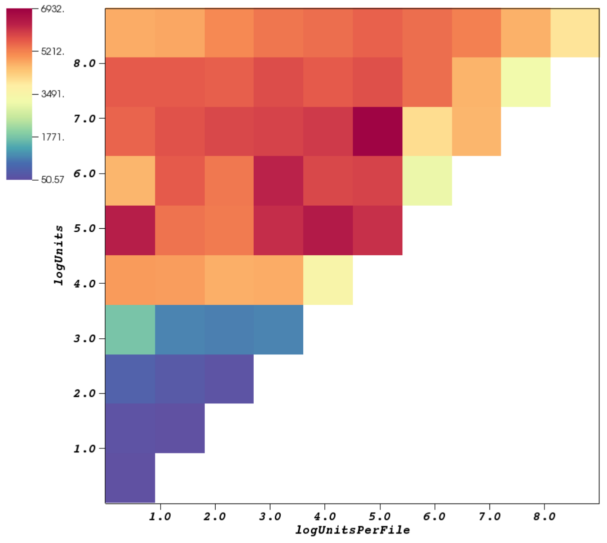
32MB vs. 32GB. The I/O sizes were chosen to represent easily testable small (32 nodes, 16 cores per node, 64KB stripes for a total of 32MB) and large (32GB as the largest power of two file size able to be written by a single core) tests. For the results shown in this Section (Figures 3 and 4) the reduction operator used for the 2D histogram is the maximum. Each bin is then colored by the maximum throughput of any I/O in that bin. For each test we create an additional figure encapsulating the efficiency of the I/O models, and is calculated as the throughput per node hour of each I/O model. Node hour was chosen as this is the metric by which Blue Waters project allocations are charged.

Figure 3 shows the results of our small I/O test. Figure 3(a) adheres to expectations; parallelizing I/O increases I/O throughput proportionally. However, even at such small scale testing file per process I/O is already not the model with the highest throughput. Instead at least a small level of aggregation on disk is required to achieve this. While this supports the common notion that with little effort high I/O throughput may be achieved utilizing a file per process I/O pattern, Figure 3(b) supports parallel I/O best practices. In that Figure, we can see that raw throughput does not correspond to efficiency. Interestingly, what appears most efficient corresponds closely to common HPC center advice: aggregate to the node level. The sharp transition in Figure 3(b) from low to high efficiency of I/O corresponds to the point when multiple cores are writing to each of the eight OSTs.

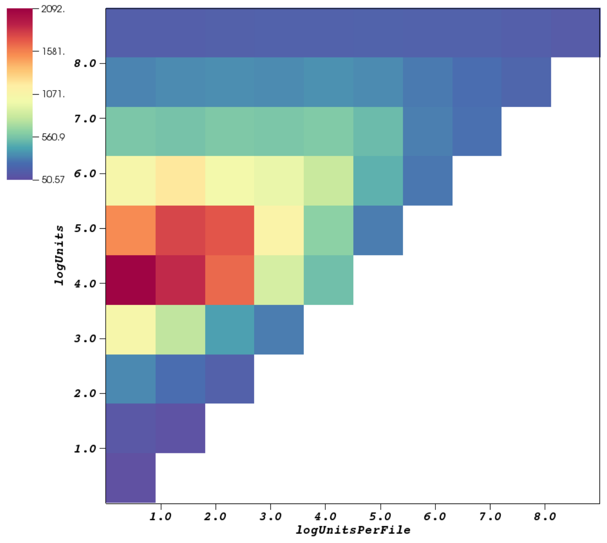
Figure 4 shows the results of our large I/O test. With the larger test, we are not able to achieve even near maximal throughput with file per process. Among the maximal raw throughputs are now the file per node I/O model, although additional aggregation on disk in some instances does indeed result in higher raw throughput. Figure 4(b) highlights the critical importance of correct tuning for parallel I/O. In that Figure there are few efficient I/O models, but these are well aligned with the models offering good throughput. In both cases and to our knowledge, for the first time, we are able to provide I/O benchmark results that not only provide advice on how to optimize an application's I/O (we provide an upper bound on what to expect from the development effort to aggregate I/O) but also evidence that validates in a clear and accessible way the best practices provided by HPC centers.

VI. Conclusion and Future Work

In this paper we presented a new classification for parallel I/O that in a single context provides an understandable display of performance that may help optimize an application's I/O but also provides evidence of the appropriateness of HPC centers' best practices. This work is a first step and is rich with potential directions for future effort. First, this new setting provides additional opportunities for filesystem analysis. Simply changing the reduction operator for the bins of the 2D histogram may provide additional insights. For



(a) 32GB I/O throughput.



(b) 32GB I/O efficiency.

Figure 4: 2D histograms of writing 32GB on various node/core counts from 1 to 512 cores across 32 nodes. Each bin is a maximum of throughput (a) or throughput per node hour (b) and represents an I/O model or set of models.

example, a cursory look at using variation as the reduction operator shows outliers in each of our test cases; we need to further analyze the relationships among I/O models falling in the same bin.

Additionally, much further testing is necessary. It is unclear whether the results presented in this paper are possible to reproduce with alternate APIs as the inclusion of elements such as metadata increase difficulty in ensuring stripe aligned I/O. Secondly, we must evaluate the effect of usual system noise in a more realistic test setting, e.g. on Blue Waters. Also as testing amounts to an application specific parameter sweep, we need a framework that makes such testing viable more generally for an HPC center.

Acknowledgment

This work is funded by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. We would like to thank Kalyana Chadalavada for many thought provoking conversations regarding I/O classifications as well as Mark Van Moer for helping work through possible visual representations.

References

[1] K. Chadalavada and R. Sisneros, "Analysis of the blue waters file system architecture for application i/o performance," in *Cray User Group Meeting (CUG 2013), Napa, CA*, 2013.

[2] "I/O and Lustre usage." [Online]. Available: https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips#io-best-practices

[3] "Optimizing I/O performance on the Lustre file system." [Online]. Available: http://www.nersc.gov/users/storage-and-file-systems/optimizing-io-performance-for-lustre

[4] R. Thakur, E. Lusk, and W. Gropp, "Users guide for romio: A high-performance, portable mpi-io implementation," Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, Tech. Rep., 1997.

[5] "IOR: interleaved or random hpc benchmark." [Online]. Available: https://github.com/chaos/ior

[6] H. Shan and J. Shalf, "Using IOR to analyze the I/O performance for HPC platforms," in *Cray Users Group Meeting (CUG) 2007*, Seattle, Washington, May 2007.

[7] P. Wauteleta and P. Kestener, "Parallel IO performance and scalability study on the PRACE CURIE supercomputer," Partnership For Advanced Computing in Europe (PRACE), Tech. Rep., September 2009.

[8] R. Henschel, S. Simms, D. Hancock, S. Michael, T. Johnson, N. Heald, T. William, D. Berry, M. Allen, R. Knepper, M. Davy, M. Link, and C. A. Stewart, "Demonstrating lustre over a 100gbps wide area network of 3,500km," Salt Lake City, Utah, 11/2012 2012.