Characterizing and Approximating I/O Behavior of HDF5 Applications

Neeraj Rajesh, Anthony Kougkas, Xian-He Sun
Illinois Institute of Technology
Chicago, USA
nrajesh@hawk.iit.edu,{akougkas,sun}@iit.edu

Gerd Heber
The HDF Group
Urbana-Champaign, USA
gheber@hdfgroup.org

1 EXTENDED ABSTRACT

With a suitable analysis and an understanding of the application I/O behavior, valuable insights into the causes of poor performance can be derived. In this work, we aim to characterize the I/O of an application and using this understanding try to answer the following questions:

a) Can we decompose an application profile into a simple sequence or "genome" sequence that describes the application?
b) Can we compare these different "genomes" using a simple similarity metric?
c) Can we compare the I/O characteristics of 2 or more applications using their "genomes"?,
d) Can we use a simpler mechanism to recreate the I/O behavior of the application?
e) Can we infer useful information about the profiled application's behavior?

To answer the above questions, we use the Darshan[4] profiler to analyze and classify these applications using a simple cosine similarity metric. Our first goal is to understand the predictive power and the limitations of the chosen similarity metric for understanding the I/O of an application. Our second goal is to approximate the I/O of a target application using a similar but a simpler proxy application. The systematic approximation of the I/O of an application might yield insight into poor application performance and clues as to how we can improve it.

2 METHODOLOGY

We profile the I/O behavior of an application using Darshan. From the performance counters we decompose them to feature vectors. These feature vectors essentially represent the I/O behavior of the application. Using them, we can compare the I/O behaviors of the other application using cosine similarity [3]. To understand the performance of these applications and the metric we compared them to applications[1, 2] whose I/O behavior was already known. We then attempted to emulate the I/O behavior of these applications using a workload generator. This is a simpler mechanism that can be used to recreate the I/O behavior of an application without needing the input application.

(1) **Extraction of Signatures** The Darshan profile has a number of different I/O modules, POSIX, MPI-I/O, HDF5 (H5F + H5D), STDIO, and more. We have limited it to POSIX, MPI-I/O, and HDF5. The Darshan profile of an application includes a set of *record IDs* whose number is equal to the number of files accessed by the application plus the number of HDF5 datasets accessed. We use the cosine similarity to organize all record IDs in equivalence classes, where each equivalence class represents a different kind of I/O behavior in the application. We extract one Record ID from each class to form an *application signature*. In the creation of the HDF5 application signature, we combine POSIX and MPI-I/O (if PHDF5 is used) performance counters for a given HDF5 file. Currently, we have only studied HDF5 applications with application signatures that consist of a single equivalence class.

(2) **Application Comparison** Applications are compared via the cosine similarity of their application signatures. Prior to comparison, the signatures need to be converted into *normalized feature vectors*. The first step is to convert the signature to a feature vector. The individual components of the Darshan record can be represented as key/value pairs in a dictionary, this ensures the formation of consistent feature vectors across applications. We create feature vectors such that values at every index of the vector correspond to the same performance counter across applications. Now the feature vectors are ready to be normalized

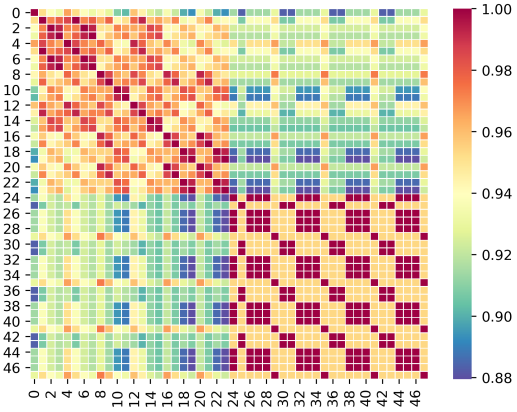
Normalization The main purpose of normalization is to prevent one or a small set of features to dominate and distort the similarity calculation. Two types of normalization were considered in this project:

- **Feature-Range Normalization**. We vertically stack the application signatures and normalize each column vector of the newly formed application signature matrix between 0 and 1 through a min max normalization technique.
- **Pairwise Normalization**. We apply max normalization on the two values of the Feature Vectors at an index.

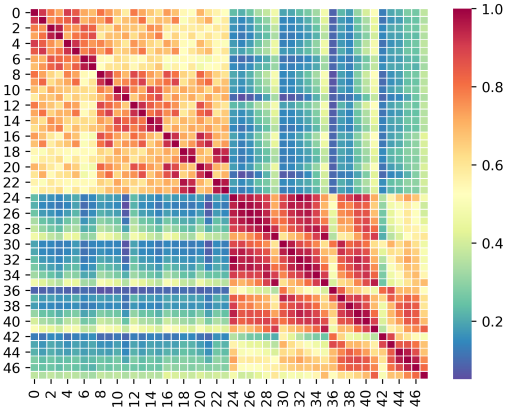
(3) **Create an Approximation** This is done using a *genetic algorithm*. It takes n random configurations of the workload generator. It then creates a population of size n, these individuals are evaluated using cosine similarity to the target application. They are selected (tournament style), crossed over (2 point cross-mate), mutated (on a certain probability), and subsequently generate a certain number of new individuals for a specified number of generations. It then tries to find the most (cosine) similar configuration for the given input application profile.

(4) **Generate Workloads** These workload generators have different I/O based on different configurations. These workload generators and a configuration describing an I/O pattern would be used as a proxy application for a target application. Three workload generators were used:

- IOR, which can emulate simple I/O behavior
- H5Perf, which can emulate simple HDF5 specific behaviors



(a) Pairwise Normalization



(b) Feature Range Normalization

Figure 1: Heatmap of Cosine Similarity of applications with-different I/O behaviors

- Fio, which can emulate a variety of complex I/O behaviors using different I/O engines and can support so-called job files

Workload generators are used for the search algorithm, as *approximators*. To assess and quantify the approximation limits, as *resonators*. The run time of the genetic algorithm can be reduced through parallel evaluation and memoization of previous runs.

(5) **Quantify limitations** The aim is to find how close an application can get to the target application and figure out how effective the workload generator can be. To give us an understanding for how effective searching for a workload generator configuration can be.

3 ANALYSIS

a) **Application Profiling**: While profiling the application we notice that the most information we can get about the I/O is limited by the performance counters available in Darshan. Also, currently we have made the assumption that an application has only one equivalence class in its signature. This however, is not the case in reality and applications can have more than one equivalence class in its signature. One way to deal with this is to create a collection of application signatures of all the different unique equivalence classes. We can now go about profiling, approximating and further analysing each of the different signatures and combine them to get an understanding and a proxy application of the profiled application.

b) **Application Analysis** (Figure 1(a), 1(b)): In order to analyze and understand different unknown application it is useful to have a collection of known applications that we understand as completely as possible. Then using the understanding of the known applications we can attempt to understand how similar the unknown application is to our collection of known ones. This should give us an understanding of the unknown applications. We see that the more context, in terms of known applications, we have of the I/O of an application we more we can infer about an unknown information. Also, once again the understanding of the application is bound by how the I/O of an application has been described by the profiler, in our case Darshan.

c) **Quantifying an Application** (as referenced in Figure: 3, Figure: 4 in poster): The workload generator has certain limitations when emulating workloads not native to it. This is why it helps to have a collection of workload generators so that we can map an application to a workload generator that can emulate it as close as possible. Hence, the I/O emulated by a Workload Generator is at best an approximation of the I/O behavior of the application. It is hard to get a perfect match.

d) **Application Approximation**: The Approximations are a lot closer to the profiled application when the approximator has information about total I/O size and block size information as it prunes the search space of the genetic algorithm. It helps to have multiple solutions for an approximation that way different configurations to a workload generator can give a good enough approximation but pruning it down can help increase its accuracy by simply reducing the vast search space. This must be done carefully so as to not over prune it.

4 CONCLUSION

When profiling an applications I/O, the more variety of application profiles available the more accurate it is to compare different I/O behavior types. While the application can be approximated, there are limitations based on the available workload generator for the kind of I/O patterns it can possibly do. Emulating these patterns also depends on the different Performance Counters that can aptly describe various aspects of I/O Behavior.

REFERENCES

[1] Joe Glenski David Richards. [n. d.]. Best Practices for Using Proxy Apps as Benchmarks. Available athttps://www.youtube.com/watch?v=JpdvGz-2LKI and http://ideas-productivity.org/wordpress/wp-content/uploads/2020/04/webinar039-bpproxyapps.pdf April 15, 2020.

[2] James Dickson, Steven Wright, Satheesh Maheswaran, Andy Herdman, Mark C Miller, and Stephen Jarvis. 2016. Replicating HPC I/O workloads with proxy applications. In *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*. IEEE, 13–18.

[3] Thomas Kistler and Michael Franz. 1998. Computing the similarity of profiling data. In *Workshop on Profile and Feedback-Directed Optimization*. Citeseer.

[4] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. 2016. Modular hpc i/o characterization with darshan. In *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 9–17.