**Abstract**

By characterizing the I/O behavior of applications, we try to answer the following questions:

- Can we decompose an application profile into a simple sequence or “genome” sequence that describes the application?
- Can we compare different “genomes” using a simple similarity metric?
- Using a simple similarity metric can we,
  - Compare the I/O characteristics of 2 or more applications using their “genomes”?
  - Can we use a simpler mechanism to recreate the I/O behavior of the application?
- What can we infer about the original application’s behavior?

**Motivation**

From suitable analysis and understanding of application I/O behavior, we can derive insight and clues into the causes of poor performance. In this effort, we use the Darshan profiler, we analyze and classify the cosine and other similarity metrics. Our first goal is to understand the predictive power and limitations of similarity for understanding the I/O of an application. Our second goal is to approximate the I/O using the cosine and other similarity metrics. 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 the corresponding features 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.

**Create an Approximation**

- This is done using a genetic algorithm.
- It takes n random configurations of the workload generator.
- It creates a population of size n, individuals are evaluated using the cosine similarity to the target application.
- They are selected (tournamont 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.

**Generate Workloads**

- These workload generators have different I/O based on different configurations.
- The workload generator along with a specific configuration describes an I/O pattern that would be used as a proxy application for the target application.
- Three workload generators were used:
  - IOR, which can emulate simple I/O behavior
  - H5Perf, which can emulate simple HDF5 specific behaviors
  - Fix, which can emulate a variety of complex I/O behaviors using different I/O engines and which supports so-called job file
- Workload generators are used in two roles:
  - For the search algorithm, as an approximator
  - To assess and quantify the approximation limits, as renovators
- The run time of the genetic algorithm can be reduced through parallel evaluation and memoization of previous runs.

**Conclusions**

- When trying to profile the applications
  - Our information about the I/O behavior of an application is limited by the performance counters available in the Darshan profiler.
  - An application can have more than one equivalence class in its signature.

When analyzing an unknown application

- A collection of known workloads might help to “position” an unknown application.
- The more context we have the better our understanding

With quantifying the Workload Generators

- The workload generator has certain limitations with emulating workloads not native to it.
- 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.

When making the approximation

- The approximations are a close hit when it has more information about the I/O being approximated.

- H5Perf to approximate HDF5 I/O Tests

- When profiling an application’s 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 on the kind of I/O patterns it can possibly do.

- Extracting of Signatures

**Extraction of Signatures**

- To create 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.

**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 requires the formation of consistent feature vectors across applications.
- We create feature vectors such that the 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 the corresponding features 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.

**References**