Intellidrive: Improving Disk Drive Read Access with Machine Learning Algorithms

Peter Gebhard Philip Joseph Vance Thornton
University of Illinois at Urbana-Champaign
Department of Computer Science

1 Introduction

In the unwavering demand for ever faster computing, we have grown accustomed to a relatively stagnant rate of improvement in hard disk drive access times. Meanwhile, advances in CPU, GPU, and RAM technologies continue to push the limits of performance. It is from these observations that we are focusing on a method to alleviate the disk drive from its mechanical limitations and improve it's perceived performance through more sophisticated caching.

The typical approach to solving data latency issues is to build a caching store as a new layer between the requester and the requestee. To improve disk drive access performance, modern hard disk drives now commonly include a read/write disk buffer that allows the disk drive to cache sectors ahead of (and possibly behind) a requested sector read. This same disk buffer can also be used to queue incoming write requests so the system's OS can continue other processing. So while most system disks already contain a disk buffer, we feel that disk read performance can be improved by moving beyond a simplistic read-ahead approach and instead applying a machine learning algorithm. Our approach will not focus on any significant changes to how write requests are queued and scheduled.

In current approaches much of the responsibility for data access performance is placed on the application developer. Decisions regarding the use of files and the organization of data within files can have a significant impact on data access performance. Choosing an appropriate data organization can be a difficult task that is even more difficult when it is not known in advance what the common access patterns will be. Current approaches attempt to ease this burden by providing caching based on simple assumptions such as that data that was accessed recently is more likely to be accessed again soon and that data that is near data that is being read in a file is likely to be read soon. Existing file system implementations attempt to reorganize data so that files are stored sequentially on the disk. Whether or not these assumptions about disk access patterns and appropriate block organization are correct is highly application dependent and may change over time.

Our approach has the potential to be better than current approaches because the access pattern prediction utilizes more sophisticated models that are based on access history and adapt over time. By providing dynamic block reorganization we will also hopefully reduce or eliminate the need for application developers to consider the performance implications of data organization as data organization will adapt over time based on usage.

The anticipated result of this project is that disk read performance will be improved for tasks that require repeated random I/O for sets of related data. Improving write performance is not a goal of this project, however a potential benefit of intelligent block reorganization is that random writes can be replaced with sequential writes which are later reorganized based on access patterns.

2 Timeline

Our proposed development timeline is shown below broken into various phases.

2/15 - 2/28

- Implementation of memory based Linux block device
- Implementation of disk based Linux block device
- DiskSim evaluation and prototyping
- Evaluation of machine learning algorithms
- Evaluation of existing implementations of selected machine learning algorithm

3/01 - 3/14

- Implementation of block access prediction test
- Implementation of intelligent pre-caching

- Implementation of intelligent block re-organization
- Integration of intelligent pre-caching and block re-organization with Linux block device
- Integration of intelligent pre-caching and block re-organization with DiskSim

3/15 - 3/31

- Definition of test work loads
- Execution of test work loads using memory based implementation
- Execution of test work loads using intelligent pre-caching
- Execution of test work loads using intelligent block reorganization
- Execution of test work loads using both intelligent pre-caching and block reorganization

4/01

- Midterm report

4/01 - 4/23

- Refinement and enhancement of intelligent pre-caching and block reorganization implementations

4/24

- Project presentation

4/24 - 5/10

- Final performance evaluation

5/11

- Final report and presentation

3 Anticipated Results

We expect our final project demo to perform better for some tasks, and perform as well or maybe worse for other tasks. Based on other research papers that we have read, this seems to be the norm. As long as we can show more of an improvement than deterioration, then we believe our project should be a success. Also, if our approach is only good for specific activities (i.e., database accesses), then we could change the scope of our project from being a general purpose program to a specialized program that could be used specifically for those activities.

We have concerns that overhead of our system may severely worsen the results of our project. We don't want accesses to take too long and we don't want there to be a lot of data stored per block. Many processes are CPU intensive, and our program might slow down the CPU a little more. With the rate CPU speed increases versus disk speed, it is probably better though for the CPU to take a hit. One thing, that may be hard in achieving favorable results, is balancing the benefits that you get with the amount of information that you want to store.

One other setback that concerns us is that we might find that the OS cache gets in our way. We have to make sure that we are working with a clean cache in order to accurately measure our performance. A virtual machine may be good for this since we can take snapshots and load those up. DiskSim [1] might also be good for this.

We don't intend to have any improvements in write performance. The knowledge of how blocks of data are read off of the hard drive, however, may provide potential side benefits to future work trying to improve write performance.

4 Evaluation Plan

Depending on what frameworks we decide to employ, our path for evaluating our project will vary. As the implementation of our algorithms will tend to focus on improving read access times, we will logically want to focus our attention on tasks that involve a large amount of disk reads. In general, the path for evaluation will be to run different I/O intensive tasks with and without our changes. We will measure the amount of disk block reads, the order in which the blocks are read, and the general amount of time to perform the task. We will use the order of the block reads to determine if and how we might be able to improve our algorithms. Measurements will be taken during our tests using tools such as DiskSim and OS-level performance-monitoring software.

Some of the different types of I/O intensive tasks that we are currently considering include database usage, video and audio encoding/decoding, video and audio playback, software development activities (compilation, linking, loading of IDEs, etc.), and general computer usage (web browsing, email checking, word processing, etc).

References

[1] JOHN S. BUCY, JIRI SCHINDLER, S. W. S. G. R. G., AND CONTRIBUTORS. The DiskSim Simulation Environment Version 4.0 Reference Manual.