Enhancing the Accuracy of Synthetic File System Workloads

Salam Farhat – Advisor: Dr. Gregory Simco

Introduction

Operating system benchmark suites consist of tools that aide in file system performance measurement and analysis (Agrawal, Dusseau, & Dusseau, 2008, Traeger, Zadok, Joukov, & Wright, 2008). Integral to benchmark suites are workloads that exercise the target systems in repeatable scenarios that are indicative of common application execution. One workload generation methodology involves the study of operating system access patterns by a set of applications that exhibit functions common to production software. The patterns are then incorporated into a tool that reproduces a typical application's function into representative synthetic workloads in the context of benchmark suites (Agrawal, Dusseau, & Dusseau, 2008). Thus the generated workload reproduces primary and secondary storage access with the goal of providing equivalent execution sequences through application programming interface (API) or systems calls indicative of an application's interaction with the operating system (Roselli, Lorch, & Anderson, 2000).

Problem

Current synthetic benchmarks do not generate representative workloads due to the assumption that mimicking an application's API calls is sufficient to reproduce the application's workload (Traeger, Zadok, Joukov, & Wright, 2008). This oversimplified assumption fails to account for the different execution paths that consist of lower level operating system function calls (Agrawal, Dusseau, & Dusseau, 2008, Joukov, Wong, & Zadok, 2005). An API call can have more than one execution path with a significant difference in performance. As a result two workloads with the same API calls but different execution paths will have a different performance footprint that is dependent on the function calls. An example of an API call that has multiple execution paths is the read API call. The read API call can have an execution path that consists of function calls that reads the file from primary storage, or an execution path that consists of function calls that reads the file from disk whose performance depends on whether the file is fragmented or not.

Goal

The goal of this work is to advance file system benchmarking by providing a tool to generate a synthetic workload that is representative of real applications. This is accomplished by considering how the application's system calls are satisfied by the lower function calls the workload would exercise the hardware in a more accurate manner that in turn allows for system designers to better enhance their systems (Agrawal, Dusseau, & Dusseau, 2008, Zhang, Sivasubramaniam, Franke, & Gautam, 2004).

This work will extend the synthetic disk workload generation process presented in Tarasov et al. (2012), but instead of tracing disk calls it will trace the virtual file system calls that include file system calls that are satisfied from primary storage and secondary storage. From the traces input in the form of configuration files for existing file system benchmarks such as FileBench will be generated.

To validate the work the generated synthetic workload will be compared against a workload it is trying to mimic. This workload can be any workload such as a live application or generated by a synthetic workload such as Postmark. The synthetic workload's accuracy can be measured by how close it mimics a workload application by comparing the distributions of the API calls and function calls for both traces. For each parameter the percentage difference between both workloads is an indication of how closely the synthetic workload matches the workload it's trying to mimic.

Approach

This work will extend the methodology used in synthetic disk workloads provided in Tarasov et al. (2012) and apply it to file system workload generation. So first a benchmark such as Postmark would be chosen to generate the designated live workload to be imitated synthetically. Tracefs would be used to capture VFS level traces (Aranya, Wright, & Zadok, 2004) and from these traces the configuration files that are used to run the FileBench benchmark would be generated. The synthetic workload generated by FileBench would be traced and validated against the traces from the live workload.

Tracefs is a file system driver built as a kernel module that implements file system operations. When a file system call passes through tracefs it is recorded along with its parameters along with the timestamps and process id. The traces can be configured to include additional information such as owner id and session id. One problem with tracefs is that it does not work on current Linux kernels. The latest version of tracefs was built for the Linux kernel version 2.6.17.13 that has since been modified greatly. So the first step in the approach is to port tracefs to a current Linux kernel.

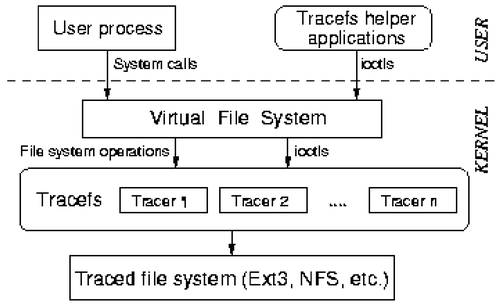
The second step of this process is to install Postmark and configure it to run the live workload that will be traced using tracefs. Postmark can be run under different configurations to ensure our process can create representative synthetic workloads on varied live workloads. Some postmark configuration parameters include the initial number of files to create, the range for file sizes, the number of read, write, create, or append transactions, and the average read and write sizes.

From the Postmark traces a statistical model similar to the one presented in Tarasov et al. (2012) is generated. The statistical model was presented earlier and basically consists of creating an n-dimensional matrix where each dimension consisted of a feature recorded in the trace such as VFS calls and their parameters. The values of each cell represent the value of the parameter or VFS call.

The work in Tarasov et al. (2012) realized that no benchmark is capable of modeling the variation within one live workload. An example of workload variation is if the live workload begins with a high number of reads and ends with a high volume of writes. Current synthetic benchmarks do not provide the capability to vary workloads and can only provide the configured overall API calls across the entire workload duration.

The authors of Tarasov et al. (2012) divided the traces into 20 second intervals and created a statistical model for every interval. Then for every statistical model a configuration file was created and the benchmarks were configured to run these configuration files in the provided order for the same interval that is 20 seconds. The 20 second interval was determined experimentally starting with larger intervals and systematically decreasing it until the desired accuracy is achieved. In this work the same approach can be used to determine the largest possible interval that would still enable the generation of representative workloads.

Tracefs is also used to collect the traces generated by FileBench and the statistical model are generated for the synthetic workloads for every 20 second interval. Each interval is then compared to the corresponding interval of the live workload. As discussed in the goal the root mean square (RMS) and the maximum distance are calculated for every parameter. The lower the value the more closely the two workloads resemble each other.



References

Agrawal, N., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2008). Towards realistic file-system benchmarks with CodeMRI, ACM SIGMETRICS Performance Evaluation Review, 36(2), 52-57.

Aranya, A., Wright, C.P., & Zadok, E. (2004).Tracefs: A File System to Trace Them All. Proceedings of the 3rd USENIX Conference on File and Storage Technologies, 129-145

Roselli, D., Lorch, J.R. & Anderson, T.E.(2000). A Comparison of File System Workloads. Proceedings of the annual conference on USENIX Annual Technical.

Tarasov, V., Kumar, S., Hildebrand, D., Povnzner, A., Kuenning, G., & Zadok, E. (2012). Extracting Flexible, Replayable Models from Large Block Traces. FAST 12.

Traeger, A., Zadok, E., Joukov, N., & Wright, C.P.(2008). A Nine Year Study of File System and Storage Benchmarking. ACM transactions on storage, 4(2).

Zhang, J., Sivasubramaniam, A., Franke, H., & Gautam, N. (2004). Synthesizing Representative I/O Workloads for TPC-H. Proceedings of the 10th International Symposium on High Performance Computer Architecture, 142-152.