Findings from the First Annual File and Storage Systems Benchmarking Workshop

Avishay Traeger[1], Erez Zadok[1], Ethan L. Miller[2], and Darrell D. E. Long[2]
[1]*Stony Brook University*
[2]*University of California, Santa Cruz*

A growing consensus in the community of file and storage system researchers and practitioners believe that the quality of benchmarking must be improved significantly. We have found that there is often too little scientific methodology or statistical rigor behind current benchmarking, which is largely done ad-hoc. In response, with the goal of improving the quality of performance evaluation in the field, we held the Storage and File Systems Benchmarking Workshop on May 19, 2008 at the University of California Santa Cruz. It was sponsored by the Storage Systems Research Center (SSRC, *www.ssrc.ucsc.edu*).

This workshop brought together top researchers and practitioners from industry and academia, representing all levels of the storage stack, along with statisticians and other interested parties. The main goals of the workshop were to educate everyone on the problems at hand, and discuss possible solutions. Participants presented relevant topics, and there was much interaction and discussion.

The goal of this effort is improving the scientific and statistical methodologies used. This goal requires little research in the field, but does require educating both those who conduct performance evaluations and those who analyze results. It also requires program committees and reviewers to raise the bar on the quality of performance evaluations in accepted papers. A longer-term goal is to have computer scientists embrace the rigor of the other sciences. It is essential to be able to validate the results of others. Without it, it is meaningless to compare the performance of two systems. All presentations and slides are available at *www.ssrc.ucsc.edu/wikis/ssrc/BenchmarkingWorkshop08/*.

Why File and Storage System Benchmarking is Difficult

Erez Zadok, the workshop's chair, began with an overview of the storage stack, some complexities that make benchmarking these systems a difficult task, and some examples of poor benchmarking practices. Some of the factors contributing to the complexity are:

- **Storage variety:** storage does not consist only of a single local hard drive. Other types include Logical Volume Managers (LVMs), RAID, Network-Attached Storage (NAS), Storage Area Networks (SANs), flash, object storage, virtualization, etc.
- **File system variety:** many types of file systems exist. Those operating on a local disk can use different data structures, logging infrastructures, and other features such as encryption or compression. Network file systems behave differently from local ones because of cache effects and network latencies. They are very common today and distributed file systems are becoming more so.
- **Operating system variety:** several operating systems exist, each with different behaviors. In addition, running OSs in virtual machines is becoming more common. Finally, even the same OS will behave very differently depending on the configuration.
- **The workload:** user activity and access patterns are difficult to accurately characterize and recreate.

- **Asynchronous activity:** other processes and kernel threads may also interact with the storage stack and change the system's behavior.
- **Caches:** operating system caches at various levels, as well as disk caches, can contain recently accessed data and metadata, which can change the behavior of the workload.

The Current State of File and Storage System Benchmarking

The next presentation was from Avishay Traeger (Stony Brook University), summarizing his recent article [6]. The article surveys the benchmarks and methodologies that were used in file and storage system papers from SOSP, OSDI, FAST, and USENIX between 1999 and 2007, and included 415 benchmarks from 106 papers. It also looks at how testbeds and results were presented, and suggests better benchmarking practices. Some of the findings were that approximately 47% of papers did not specify how many runs were performed, and more than 28% of the benchmarks ran for less than one minute. In addition, only about 45% of the papers had some indication of variance (standard deviation or confidence intervals).

Current Benchmarks

The benchmarks presented at the workshop were IOzone [2] and SPECsfs [5] (both presented by Don Capps of NetApp), and FileBench [3] (developed by VMware and Sun Microsystems, presented by Spencer Shepler of Sun). In contrast to benchmarks that are generally used, these benchmarks provide important improvements. SPECsfs presents new techniques for scalable workload generation. IOzone and FileBench can create a variety of user-specified workloads, which may help to reduce the number of ad-hoc benchmarks that are created and used. Ad-hoc benchmarks, are generally small programs that are written for in-house use. Using popular tools in favor of ad-hoc micro-benchmarks can aid in reproducing and comparing results, as now only the workload specifications need to be reported, rather than the source for the entire benchmark. In addition, we would expect that the more popular tools will have fewer bugs and operate more correctly.

IOzone is a portable open-source file system benchmarking tool that can produce a wide variety of I/O, and more recently, metadata workloads. It can produce single or multiple execution threads, and even run on multiple nodes. An interesting feature of IOzone is its use of telemetry files. IOzone can replicate I/O operations based on a file containing $\langle$*byte offset*, *size of transfer*, *compute delay*$\rangle$ triplets, so that it can provide benchmark results from system call traces. IOzone has been downloaded millions of times, and is the first result on Google when searching for "file system benchmark." Surprisingly, IOzone was not used in any of the conference papers surveyed by Traeger et al. In fact, many researchers publishing in the surveyed conferences have written their own benchmarks that produce workloads that IOzone can easily produce. We can only speculate about the reason for this phenomenon at this point, as we have no hard data, but we believe that this may be another indication of poor benchmarking practices in the file and storage system community.

SPECsfs is a file server benchmark that measures both throughput and response time. SPECsfs was originally created to test NFS servers. The latest version, SPECsfs2008, supports CIFS in addition to NFS. The major changes to the NFS portion of the benchmark since version 3.0 are updated I/O size distributions, a new operation mix, and the dropping of UDP and NFSv2 support. The CIFS portion is rather different, using a Hidden Markov Model driven by traces to generate the workload, rather than a pre-defined operation mix. The workloads for both NFS and CIFS are now based on data from many real customers. It is important to note that SPECsfs2008 cannot be used to compare NFS and CIFS servers.

An interesting point that was brought up is that the NFSv4 protocol depends much more on the clients' behavior than previous versions. To benchmark a complete NFSv4 system, the client's behavior should

be taken into account. This means that the method that SPECsfs uses for benchmarking NFSv3 systems would not be applicable to NFSv4 (the benchmark crafts its own RPC packets). Any current benchmark that uses the POSIX interface can send requests to an NFSv4 server via a real client, thereby taking the client's behavior into account. However, it is up to the user to define what constitutes an appropriate file server workload for their system; for that, configurable workload generators such as IOzone and FileBench can be used. In the future, we hope the community will define one or more standard file server workloads that are generally applicable, and revise them periodically. Of course, additional benchmarks may be used as well to provide a clear picture of the system's performance characteristics.

At times benchmarking applications can be a very difficult task. For example, properly running up a TPC-C database benchmark is very expensive and may require several months of time to set up and run. In addition, we do not currently know how to extrapolate micro-benchmark results to reflect the performance of real applications. Therefore, we need to use macro-benchmarks, which more closely represent the applications themselves, and build a portfolio of workload-specific benchmarks. FileBench was developed as a method of accurately representing more complex file-based applications, so that the performance impact of a file system or storage layer can be properly characterized for specific workload types. It uses a synthetic workload model to accurately represent the workload and application stack, including the process model, the I/O types, synchronous I/Os, and most importantly the interlocking between I/Os. It also provides the framework for operating on statistical hierarchies of file system trees, and high level file system objects, including create/delete, traverse directory, read/write etc.

Short-term Goals for Benchmarks

We realize that creating a perfect solution will involve much research and community involvement. However, there are steps that we can take now to make benchmarks more accurate and help facilitate comparable and reproducible results. In terms of accuracy, the benchmark should use accurate timing in measuring metrics. Eric Anderson (HP Labs) also pointed out the importance of accurate timing in issuing file system and I/O requests. It should also be a simple, easy-to-understand workload. This helps ensure accuracy, and also helps in understanding the results and their implications. The benchmark should also accurately depict a real world scenario if its goal is to do so. How to measure this accuracy, however, is an open problem. Finally, open source benchmarks promote openness and allow for more people to inspect the code for correctness. Of course, the code should not be modified so that results remain comparable.

In terms of comparable and reproducible results, the benchmark should have three main qualities. First, they should be scalable. Benchmarks may properly exercise the system at one point in time, but as systems become faster, the benchmark may no longer be appropriate. For example, a common benchmark is measuring the time required to compile some source code (as in the Andrew benchmark). However, source code that was used for benchmarks several years ago would fit in a modern system's cache and therefore would not adequately exercise the storage subsystem. Second, benchmarks should have few dependencies on libraries and the OS. For example, the Bonnie benchmark creates a random read pattern by utilizing the system's pseudo-random number generator. This causes the read pattern to change between systems, which can lead to different results due to caching, read-ahead, and disk locality. Third, it should be cheap and easy to set up, and portable, so that it can be used by a large number of people to benchmark on many systems.

Traces

Traces are logs of operations that are collected, and later replayed to generate the same workload (if done correctly). Two problems associated with traces are availability and replay method.

The availability issue is being addressed by the Storage Networking Industry Association's Input/Output Traces, Tools, and Analysis Technical Work Group (SNIA IOTTA TWG). Geoff Kuenning of Harvey Mudd College presented an overview of this working group. They have set up a repository at *http://iotta.snia.org* that seeks to archive traces in a single place using a uniform format with tools to process them. It also helps to clear up licensing issues for the traces. The preferred trace format is DataSeries, which was presented by Eric Anderson of HP Labs. DataSeries is designed for long-term storage (built-in checksums), is self-describing, and provides substantial analysis speedups and moderate space improvements. There are tools available to convert several other formats to DataSeries, and tools to analyze the trace files.

The problem of replaying traces is partly addressed by Buttress [1], a high fidelity I/O benchmark system, which was also presented by Eric Anderson. This project demonstrates the importance of accurate issue time for I/O requests, and provides a method for issuing them much more accurately than before. However, the system is very fragile, and it is easy to specify open (trace) workloads that are unachievable and get poor results. This is a difficult and important problem that will require more research.

Industry Experiences

Several attendees presented their benchmarking experiences from the industry perspective. First, VMware's Richard McDougal, Devaki Kulkarni, and Irfan Ahmad presented their experiences in benchmarking Virtual Machine (VM) environments. When benchmarking inside of a virtual machines, it is important to note that time measurements and the CPU's clock cycle counter may be distorted (generally by around 100μs). This is especially true when the CPU is fully utilized, and can be mitigated by using *ESX-TOP* which gathers CPU utilization information from the host, by using the hardware's clock cycle counter rather than the virtualized one, or by timing from the host rather than from inside the VM. For benchmarking ESX servers, they noted that simple workloads will not suffice, as servers see different I/O patterns to the same volume, or I/O from a single application being split among multiple volumes. In addition, virtual file systems are often specially optimized, and so standard benchmarks are not always sufficient.

Next, Daniel Ellard from NetApp presented their experiences in benchmarking flash SSDs. Their goal is to perform measurements on a single device and to extrapolate to estimate the performance of a large array of devices. These new devices have characteristics that differ from disks. For example, flash SSDs implement quasi-file systems, have a strange layout that is striped across several devices, have non-deterministic writes, and have drastic aging effects. NetApp uses what they call *micro-workload* benchmarks, that are somewhere between micro- and macro-benchmarks in terms of complexity. They have developed a workload generator called Biscuit. The user defines *tasks* and these are generated by Biscuit. Biscuit also supports random variables, as well as telemetry and trace files.

The next presentation was by Jeff Fuller from Microsoft, who discussed some of the benchmarking methodologies used for Windows clients and servers. They perform client application characterization to measure metrics that end-users care about, such as high-level response time. Their application-level benchmarking allows them to use the same benchmarks on different platforms and compare user experiences across platforms. In addition, application-level workloads are more portable and realistic than lower-level ones. They also take client idle time (during which much asynchronous activity happens), as well as bursts of activity.

Finally, Eric Kustarz from Sun Microsystems discussed ZFS benchmarking experiences. As ZFS is a rather complex file system, they use a large number of workloads to obtain a clear picture of its performance. While they mainly use FileBench, they also use an assortment of other benchmarks, including IOzone, Bonnie, SPECsfs, and many others. They utilize various OpenSolaris tools to locate performance problems, such as Dtrace, Lockstat, fsstat, kstat, and vmstat.

Benchmarking Guidelines

The workshop included much discussion about proper benchmarking and statistical methodologies, and we compiled a set of guidelines to consider when evaluating the performance of a file or storage system.

A performance evaluation should have clear goals. We recommend posing questions that should be answered by the evaluation, and then choosing the systems, configurations, and benchmarks to answer them. The benchmarking process consists of four steps: selecting appropriate benchmarks, running the benchmarks, analyzing the results, and reporting the results.

At this point, hypothesize on what the results should look like, and decide on the appropriate initial state of the system (contents of caches, partition locations, file system aging, etc.) and create it accurately. When choosing a benchmark, you should use it for its intended scope. For example, the Andrew benchmark should not be used as an I/O benchmark, and Postmark produces an NFS mail server workload. In addition, create new benchmarks only if existing ones do not provide the needed features or workload characteristics. Prefer to extend existing benchmark tools rather than writing new ones.

When running the benchmarks, we recommend using an automated system [4, 7] to reduce the possibility of human error and to ensure that all runs are identical. As many data points as possible should be collected so that proper statistical analysis can be performed on the results. For benchmarks with non-uniform workloads (for example, a compile benchmark), this can be done by running the benchmark multiple times. For benchmarks with uniform workloads, such as those that perform a certain number of read operations, it may be possible to take measurements at regular intervals during a single run to increase the number of data points collected. In addition, we recommend measuring the system only when it is in steady state by discarding any start-up and cool-down effects.

For quantities that are additive (e.g., time, bytes sent), the same estimate of the mean and standard deviation should be obtained whether many short runs or just a few long runs are conducted. If a stable workload is measured by dividing it into many smaller intervals, then the central limit theorem will typically apply, and thus the distribution of the mean will be approximately normal; therefore, a confidence interval for the mean can be easily constructed from estimated standard deviations, even if the distributions of the individual runs are not themselves normally distributed. When a run cannot be broken down into multiple sub-units from identical distributions, there is no guarantee about the distribution of the mean.

The results can now be analyzed. As a first check, ensure that the distribution of the results is reasonable, and see if the results match your expectations. If not, investigate and explain why. It can be useful to examine graphical summaries, such as histograms or CDFs.

When reporting results, be sure to describe precisely what was done to help others to understand the experiments and allow them to reproduce your results. This includes a complete description of the platform, the benchmark and any parameters, the source code for the system being tested, and the raw benchmark results. Of course, licensing issues may restrict the distribution of some of this information, but as much as possible should be provided. In addition, most publications limit the number of pages available, so we recommend publishing the information in an online appendix. We hope that repositories will be created for the long-term storage of such information. In addition to describing what was done, explain *why* the evaluation was done that way. This helps others to interpret the results.

To help others interpret the results, report the number of runs performed and include statistical measurements, such as standard deviations or confidence intervals, so that others can determine the accuracy of your results. If you get high standard deviations, it could be an indication that your distribution is multi-modal (which may suggest an unstable storage system); in that case, you might plot your data as a histogram and explain the modality. Quartiles may also be helpful in describing non-normal distributions, but you should have at least 30 data points before using quartiles. In some cases box-plots may be more suitable than histograms (generally when the number of data points is large). Note that standard confidence intervals (based on a normal approximation) are not appropriate for non-normal distributions.

Summary

Many interesting and important issues were discussed at this workshop, and we hope to discuss more topics next year. These include simulators, tracing technology, aging effects, and measuring power consumption. In addition, we would like to discuss how to benchmark distributed and petabyte-scale systems, as well as virtual machine technologies. We also discovered that many are not familiar with the advanced statistical methods required to properly analyze benchmark results. We hope to discuss some of these methods as well.

Longer-term research goals were also discussed. One challenge is how to accurately scale traces so that they stay relevant for longer periods of time. This is important because a trace is collected once and used for many years. However, hardware, software, and usage patterns change rapidly, making the traces outdated almost as soon as they are captured. Another challenge is how to model an application's behavior as a workload model, and how to measure the accuracy of a given model. Finally, there is a question of how to compare the results from two benchmarks where the platform was different. The answer may lie in virtual machine technology, but how to accurately do this is an open question.

This first workshop was an important step in improving the overall quality of performance evaluations in the file and storage system community. Participants raised important issues and discussed potential solutions. We hope that researchers and practitioners will educate themselves and improve the quality of their performance evaluations. Finally, we hope that reviewers will raise the standards for performance evaluations in conference and journal publications.

We have been continuing our discussions on our mailing list, and we plan to publish a more detailed set of benchmarking guidelines in the future. We have also created a file and storage system benchmarking portal at *http://fsbench.filesystems.org/*. It links to a wiki containing the agenda (including slides from the talks) and a list of attendees, subscription information for the mailing list, a Web version of the benchmarking guidelines, and other resources.

Acknowledgments

We would first like to thank all of the attendees of this first workshop, whose valuable input and enthusiasm helped make it a success, and especially to Eric Anderson, Andrew Leung, and Tim Moore for supplying us with workshop minutes, and to Eric Anderson, Don Capps, and Richard McDougall for their reviews. Thanks to Herbie Lee for his help with the statistical aspects of this article. Thanks go to the Storage Systems Research Center (SSRC) at the University of California at Santa Cruz (UCSC) for hosting and sponsoring this workshop. Some workshop organizers were sponsored in part by NSF award CCF-0621463 (HECURA).

References

[1] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan. Buttress: A toolkit for flexible and high fidelity I/O benchmarking. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pages 45–58, San Francisco, CA, March/April 2004. USENIX Association.

[2] Don Capps. IOzone filesystem benchmark. `www.iozone.org/`, July 2008.

[3] FileBench, July 2008. `www.solarisinternals.com/wiki/index.php/FileBench`.

[4] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu. Cutting corners: Workbench automation for server benchmarking. In *Proceedings of the Annual USENIX Technical Conference*, pages 241–254, Boston, MA, June 2008. USENIX Association.

[5] SPEC. SPECsfs2008. `www.spec.org/sfs2008`, July 2008.

[6] A. Traeger, N. Joukov, C. P. Wright, and E. Zadok. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage (TOS)*, 4(2):25–80, May 2008.

[7] C. P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok. Auto-pilot: A platform for system software benchmarking. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 175–187, Anaheim, CA, April 2005. USENIX Association.