CS 333 :: Meeting Notes :: File System Aging

Thinking about benchmarks

So far, we have discussed several competing storage devices/designs/techniques (e.g., interrupts vs. polling, FFS vs. LFS, HDDs vs. SSDs) and we've used a combination of very coarse performance models and our intuition to discuss our expectations for their performance. This unit is the first time we will discuss performance *evaluation*. In the theory world, we prove things about our algorithms; in the systems world, we typically evaluate the things that we build by running a series of *benchmarks*.

A benchmark's primary purpose is to serve as a standard for comparison. By running *the same workload* starting from *the same initial conditions* on *the same hardware* in *the same execution environment*, researchers/practitioners can fairly compare two competing system designs (or even different parameter choices for the same system design). *Properly* evaluating our systems is an important part of the scientific process.

In previous courses, we've evaluated algorithmic performance in terms of the best case, worst case, average case, and/or expected case. These are all useful. But what a typical "systems user" probably cares the most about is the representative case (although some companies care very much about the worst case). In other words, what can I expect my experience to be when I use this system? A big challenge in the storage community is creating a so-called *representative state* from which to conduct the system's evaluation. Think about it: a *realistic workload* is significantly less meaningful if it is run under unrealistic conditions.

Thought questions

For these questions, consider the context of a local file system (e.g., ext4 or a similar file system running on your laptop) and think about the contents stored on disk or in memory, and think about any active work that being performed on the machine.

1. What is an appropriate *initial state* on which to run a benchmark for your particular environment?
2. What is the *steady state* of the file system in your particular environment?
3. Think back to the file system designs we've covered. Do you think that the initial/steady state should be modeled as a *path* or a *point*, and why? In other words, can we simply describe the steady state of an abstract file system by looking at the values of the bytes on your disk right now, or do we need to consider the series of operations that led up to this point in time?
4. How hard is it to create an exact representation of your current laptop state? What are some of the most important aspects to recreate?

File System Aging---Increasing the Relevance of File System Benchmarks (Smith and Seltzer)

Motivation and Goals. Measuring the performance of an empty file system is (arguably) poor science. Creating a realistically aged file system is hard but necessary work if you want to make representative measurements. Smith and Seltzer create *aging tools* that capture some key aspects of real file system workloads; these tools can then be used to replay important features of those workloads to bring a system into a reproducible state that exhibits representative aging.

Methods. The authors collected *file system snapshots* over several years. They then simulated file system workloads based on the path that those snapshots describe; they also simulated reasonable operations that, by nature, cannot be captured in the snapshot state (for example, creating and deleting temporary files).

- **Ideally:** We would evaluate a system by starting from an empty disk and then we would faithfully replay a trace of real operations that occur over months/years in order to bring that file system to an almost full state.
- **Challenge:** Collecting a perfect trace is expensive and impractical.
- **Solution:** Instead of collecting an exact trace,
  1. Collect a set of file system *snapshots*.
  2. Then, given a sequence of snapshots, generate a plausible aging workload (a series of file system operations, e.g., creates, deletes, modifications, etc.) that brings the file system state of one snapshot in the sequence to the next snapshot's file system state.
  3. In addition, also simulate the creation/deletion of short-lived files that do not appear in any of the snapshots, but may exist in the transient states between two snapshots.

They authors also verify that the workloads faithfully mirror the snapshot states. In this way, they show that their tool does in fact create *representatively aged* file system states.

New term: layout score. The paper introduces the **layout score** as a way to quantify the amount of file system fragmentation.

- An individual file's layout score is the fraction of its blocks that are optimally located within the LBA space (e.g., perfectly contiguous with respect to sequential logical blocks in the file).
  - A layout score of 1 is perfect, and 0 is the worst possible score.
  - The first block in a file is not counted against the layout score.
- The aggregate layout score of all the files in the file system is used to describe the file system's overall fragmentation.

Thought questions

1. How does a file's metadata placement affect a file's layout score? Does metadata placement affect file system performance (think back to the FFS motivation/discussion in class)?
2. The layout score is used to describe the fragmentation of individual files. Does it capture the relationship between files? How might we determine which other files are "related" to a given file?
3. What aspects of a file system workload cannot inferred from a sequence of snapshots? How do the authors attempt to simulate those aspects, and is there anything that you think they are still missing?

How to Fragment Your File System (Conway et al.)

Although it contains many meaningful insights, Smith et al. describes workloads from many years ago. If our goal is to create representative "aging" in modern environments (think about smart phones usage patterns vs. 1990s PCs...), we would need to recreate Smith and Seltzer's approach: our tools would need to reflect potentially new aging patterns that are induced by new patterns of file system usage. Unfortunately snapshot collection and tool validation is expensive, at least in terms of time.

The layout score, while informative, also has some limitations. It only captures the fragmentation of an individual file's data. We may also care about metadata placement and about the relative placement of files that are related to each other.

Conway et al. created a new tool based on the Git version control system. Their tool (1) generates an aged file system and (2) measures the file system's aging progression.

Fragmentation categories Conway et al. describes different categories of fragmentation that affect the performance of typical file system workloads. Their measurement of a file system's layout is based on performing *recursive grep* operations (i.e., `grep -r`), where a file system tree is recursively traversed and each block of each file is read (both data and metadata) in breadth-first search order.

- **intrafile fragmentation** involves blocks from the same file (e.g., as measured by Smith and Selzer's layout score)
- **interfile fragmentation** involves blocks from two related files (e.g., two adjacent files in a directory)
- **metadata fragmentation** involves at least one metadata block (e.g., the distance between a block in a file's inode and the data block it describes OR the distance between two inodes describing files in the same directory)

Methods. Their aging tool works as follows:

1. Successively use `git checkout` to replay commits from a git repository, starting from the first commit.
2. After every 100 checkouts, measure the performance of recursively grepping the file system (normalized by seconds / GiB to reflect changes in the file system size)
3. Compare the aged performance against performing the same recursive grep on a "clean" file system (a "clean filesystem" is one where the data is copied to a fresh file system partition in the order that it will be grepped. In the "clean" file system, the file system can ideally place files without performing confounding updates.)

Notes: When measuring a system, it is important to *isolate* the target effects. Conway et al. wants to isolate the effects of aging, so they create an unaged baseline, and they normalize their measurement (grep cost in seconds/GiB) to remove the effects of changing file system sizes. They also perform *cold cache* measurements so that all of the costs of seeking are paid by the system.

They later ask targeted questions that they answer by, again, isolating each of the effects.

1. Do SSDs (which have different random access behaviors than HDDs) eliminate the effects of aging? They run the same experiments on an SSD.
2. Does caching eliminate the effects of aging (and if so, what is the sensitivity to different cache sizes)? They perform warm-cache tests with different amounts of memory to identify trends and see how large the effect is.
3. Does increasing disk capacity eliminate the effects of aging (with the argument being that file systems have significantly more choice for their allocations when there is excess capacity)? Using the same physical disk, they perform the experiments with different partition sizes (which effectively expands the LBA space and eliminates disk geometry effects).

The answer to all of these questions is no. Aging appears to be a problem as long as the working set exceeds the size of the cache.

Thought Questions

1. Is recursive grep actually a good workload? What are some of its limitations? Its strengths?
2. What operations might be missed by using '''git pull''' as a workload? How might you modify the tool to incorporate those operations?
3. Is the git replay workload representative of your file system usage? What does it actually capture?

More info

The Conway et al. paper was a magazine article written to distill a larger paper down to a bite-size format. For more details, the full workshop paper (and talk) is available online in the FAST 2017 proceedings. The full paper is **File Systems Fated for Senescence? Nonsense, Says Science!** by Alex Conway, Ainesh Bakshi, Yizheng Jiao, Yang Zhan, Michael A. Bender, William Jannen, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter, Jun Yuan, and Martin Farach-Colton.

Generating Realistic Impressions for File-System Benchmarking by Nitin Agrawal, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau is another important aging paper. That paper introduces a tool to synthetically age a file system (meaning the tool doesn't replay a real workload, but instead simulates a workload based on a small number of parameters). The tool has many advantages:

- A workload can be compactly encoded and shared
- There are many parameters that let you control the settings
- Reproducibility is easy

This tool, while incredibly useful, does have some limitations. If only a few parameters are used to describe a system, the precision that the system can be described is limited. There is a tradeoff between expressiveness and complexity, and this tool strikes a nice balance; it identifies the most critical file system parameters, and gives an easy-to-use way to express them. However, if your parameter-of-choice is not among those, you are out of luck.