

# TBBT: Scalable and Accurate Trace Replay for File Server Evaluation

[Extended Abstract]

Ningning Zhu, Jiawu Chen, Tzi-cker Chiueh  
Stony Brook University  
{nzhu, jiawu, chiueh}@cs.sunysb.edu

Daniel Ellard  
Sun Microsystems Laboratories  
daniel.ellard@sun.com

## Categories and Subject Descriptors

C.4 [Performance of Systems] Measurement/Modeling Techniques; D.2.8 [Software Engineering] Performance Measures

## General Terms

Performance, Measurement, Experimentation, Design

## Keywords

File system evaluation, Benchmarks, NFS, Trace play, Aging

## 1. INTRODUCTION

TBBT is a comprehensive NFS trace replay tool that creates scalable workload-specific file system benchmarks directly from NFS traces. The resulting benchmarks are able to capture the diversity and reflect the rapid evolution of file system workloads more accurately than conventional static or parameterized benchmarks.

Creating a benchmark from an NFS trace is more complicated than simply replaying the stream of calls. First, a file system is stateful and each call accesses or modifies that state, which may have an impact on the initial state of the file system for other calls. For example, a read call can be successfully executed only if the file already exists. Second, the physical layout of a file system on disk has a significant impact on its performance, and this layout is a function not only of the strategies employed by the file system but also the “age” of the file system [3], including degradation due to disk space fragmentation. Finally, because a trace could be collected on an NFS server whose performance is very different from that of the target server, it is essential that a trace replay tool be able to scale up or down the dispatch rate of NFS calls to meet specific benchmarking requirements, all without violating any inter-call dependencies.

## 2. DESIGN

Given an NFS trace, TBBT automatically detects and repairs missing operations, derives a file system image required to successfully replay the trace, initializes the test file system with that image, ages the file system to increase the accuracy of the benchmark, and replays the trace to generate a benchmark workload. Each of these procedures may be controlled via parameters specified by the user.

Copyright is held by the author/owner(s).  
*SIGMETRICS'05*, June 6–10, 2005, Banff, Alberta, Canada.  
ACM 1-59593-022-1/05/0006

This section describes how TBBT accomplishes each step.

### 2.1 Trace Transformation

The input NFS traces consist of NFS calls and responses ordered by time. The first transformation is to match all call/response pairs, to remove certain NFS-specific details, and to add some information required for trace replay. In the resulting representation, each unique file system object is represented by a unique TBBT-ID instead of the NFS filehandle or filename that appeared in the original trace.

The trace transformation also augments the trace to account for errors or omissions in the original trace (which are most frequently caused by packet loss). For example, if a file changes size between two `getattr` calls and the trace contains no calls that would change the size of that file, then TBBT may synthesize and inject `write` or `setattr` calls to account for the observed behavior. Determining where to inject these calls based on patterns observed in other parts of the workload is an open research topic.

### 2.2 Creating and Aging the File System

Before TBBT can replay a trace, it must have a file system against which the operations in the trace may be performed. Ideally the test file system is constructed from a snapshot of the file system metadata taken immediately before the trace. If no snapshot is available, TBBT can infer a functionally equivalent approximation of the original file system from information present in the trace via techniques developed for earlier trace studies [1, 2].

The shortcoming of inferring the file system structure is that it is impossible to observe information about objects that are not referenced during the trace. Most workloads we have examined have a high degree of locality of reference, which means that much of the file system is not observed. For example, during a 14-day period the EECS workload [2] only accesses about 10% of the total space used by the file system (and most of this space is accessed during any single day). In this situation, TBBT can populate the rest of the file system with objects whose sizes are chosen randomly from a distribution modeled on that of the observed objects. It is an open question whether unobserved files have different characteristics than observed files (and therefore should have different characteristics).

Another aspect of the initial file system image is its physi-

cal layout. The physical layout of file systems populated by artificial methods (i.e.; `cp -r` or `tar`) is usually markedly different from the layout that occurs when a file system is populated during the course of normal activity and therefore suffers from fragmentation and other “aging” effects [3]. Aging impacts fragmented free space, fragmented files, and declustered objects. Therefore, when building the initial file system image, TBBT optionally simulates the aging process in order to increase the realism of the benchmark.

To induce fragmentation, TBBT interleaves the append operations to a set of files, and in each append operation adds a certain number of blocks to the associated file. To counter typical file pre-allocation techniques, each append operation is performed in a separate open-close session. The expected distance between consecutive fragments of the same file increases with the total size of files that are appended concurrently. By controlling the number of files involved in interleaved appending (the *interleaving scope*) and the number of blocks in each append operation, TBBT can control the fragmentation effects of the resulting file system. Fragmented free space is created by adding and deleting artificial objects. This aging technique is fast, effective and independent of the file system implementation.

We measured the impact of this aging technique, using different interleaving scopes and numbers of blocks written per append, on the speed that files can be read from the resulting file system. In most situations, a small interleaving scope is sufficient to produce results similar to that of real aging. By increasing the interleaving scope and decreasing the append size, it is possible to create file systems with markedly degraded performance [5].

### 2.3 Trace Replay and Load Scaling

Trace replay timing cannot simply be based upon the call timestamps in the trace, but instead must take into consideration the dependencies between the calls. For example, two write calls for different files may be scheduled to overlap, but a call to write a file must not be scheduled until the response from its creation has arrived. TBBT computes the interdependencies between each call and response and uses this information to ensure that the replay obeys them.

The *true* dependencies cannot be determined from a trace (without knowledge about the application), so TBBT provides two ordering policies to decide how aggressive to be with regard to concurrency: *conservative ordering* and *FS dependency ordering*. With *conservative ordering*, a call is issued only after all calls with earlier timestamps have been issued and responses with earlier timestamps than the call have been received. With *FS dependency ordering*, the file system dependencies of each call on other earlier calls and responses in the trace are discovered via a read/write serialization algorithm. The calls are initiated as close to their original relative timestamp as possible without violating a user-specified ordering policy. In either case, the rate of replay may be adjusted by scaling the timestamp without violating dependencies.

The *conservative ordering* rule captures some of the concurrency inherent in the trace although it will not generate a workload with more concurrency than was present in the

original trace. The *FS dependency ordering* rule may introduce more concurrency than would be possible for a specific application because it assumes that the only dependencies are file system dependencies and does not attempt to simulate application-level dependencies.

## 3. RESULTS

We evaluated the NFSv3 implementation on Redhat 7.2 using both TBBT and the SPECsfs. TBBT plays the EECS trace of 10/21/2001 [2]. We tuned the SPECsfs parameters (including changing source code) to match the trace’s characteristics as closely as possible. We measured the results for workloads based on the original trace speed as well as in (6x) scaled up speed and peak speed. For the original and 6x workloads, the latency of operations for SPECsfs is much higher than that for TBBT, and when both systems are driven to their maximum speed the average operations per second for TBBT is more than 45% higher than SPECsfs [5]. For this workload, SPECsfs greatly underestimates the performance that the server can support.

## 4. CONCLUSION

TBBT is the first comprehensive NFS trace replay tool. TBBT addresses most of the issues of creating an application-specific benchmark from an NFS trace: TBBT corrects trace errors, automatically derives information about the initial file system (if necessary) which can be used to create a test-bed file system, ages the file system in a fast, repeatable, and parameterized manner, replays the trace in a manner that preserves dependencies among calls, and permits the replay rate to be scaled faster or slower than the original trace.

In addition to being a useful tool for file system researchers, perhaps the most promising application of TBBT is to use it as a site-specific benchmarking tool for comparing competing file servers using the same protocol.

## 5. REFERENCES

- [1] M. A. Blaze. NFS Tracing by Passive Network Monitoring. *Proceedings of the USENIX Winter 1992 Technical Conference*, pages 333–343, January 1992.
- [2] D. Ellard and M. Seltzer. New NFS Tracing Tools and Techniques for System Analysis. *Proceedings of the Seventeenth Annual Large Installation System Administration Conference (LISA’03)*, pages 73–85, October 2003.
- [3] K. A. Smith and M. I. Seltzer. File System Aging – Increasing the Relevance of File System Benchmarks. In *Proceedings of SIGMETRICS 1997: Measurement and Modeling of Computer Systems*, pages 203–213, June 1997.
- [4] SPEC SFS (System File Server) Benchmark. <http://www.spec.org/osg/sfs97r1/>
- [5] N. Zhu, J. Chen, T. Chiueh and D. Ellard. Scalable and Accurate Trace Replay for File Server Evaluation. Technical Report TR-153, Stony Brook University, December 2004. [www.ecs1.cs.sunysb.edu/tr/TR153.pdf](http://www.ecs1.cs.sunysb.edu/tr/TR153.pdf)