fs123: A Scalable, Read-only Network Filesystem

John K. Salmon, Michael Fenn, Mark A. Moraes, and David E. Shaw
D. E. Shaw Research, New York, NY, 10036, USA
Email: john.salmon@deshawresearch.com

Abstract

Conventional network filesystems such as NFS have significant problems of scale and availability when used to access large scientific data sets, but are still the de facto standard because of the ubiquity and convenience of the POSIX filesystem API (open, read, seek, close, ...) they provide. To address these problems, we have developed `fs123`, an open-source, network filesystem with pervasive caching that we now use in production to efficiently distribute petabytes of data and terabytes of software to thousands of geographically distributed machines.

With `fs123`, machines needing access to remote filesystems require only a single client executable, which uses the widely available FUSE (Filesystem in Userspace) library to provide conventional POSIX filesystem semantics. FUSE callbacks are fulfilled by communicating with an `fs123` server using a custom protocol layered on top of HTTP. The `fs123` protocol consists of a small number of RESTful requests encoded into URLs. A request for the attributes of a file called foo/bar.txt, for example, might look like http://example.com/filesystem/fs123/7/2/a/foo/bar.txt. The reply is a straightforward encoding of the fields of a POSIX stat structure. Metadata is transmitted in the headers of the reply, including errors and extra information required to satisfy consistency guarantees.

`fs123` servers are relatively straightforward to implement. We have three variants in production: one that exports the contents of an existing filesystem, another that exports data from millions of tar archives, and a third that performs redirection (HTTP 302 Moved replies) using a database that maps a virtual namespace to physical replicas at multiple geographic sites. Servers and clients may be run without special permissions: neither root access, nor the ability to load kernel modules is required.

`fs123` uses caching at multiple levels to achieve scalability, managing persistent client-side disk caches while also exploiting traditional HTTP proxy caches. Cooperation with kernel-level caching makes `fs123` almost as responsive as local disk-based filesystems. `fs123` filesystems are read-only from the client application's perspective, but a filesystem's contents can be changed by server-side processes. On our production servers, for example, new data archives and upgraded software packages are added continuously. Thus, `fs123`'s caches need to respond predictably to changes in server content. An `fs123` server can exert fine-grained control over caching using the HTTP Cache-Control header, which is interpreted by proxy caches in the standard way, and which can be varied in every reply. Cache-Control, along with an additional "content validator" specified in the protocol allows `fs123` to provide a well-defined consistency guarantee: after the age specified in a file's Cache-Control header (relative to the file's last modification), client-side applications accessing it will obtain the same result as server-side applications.

Layering the network transport on top of HTTP provides several advantages. First, the software can use well-established, high-quality libraries. The existing `fs123` servers and client are written in C++, using libcurl and libevent. Running on a modern x86 machine, the servers are easily capable of saturating a 40Gbps link, though such bandwidth is almost never needed because of the pervasive caching. Second, an installation can use third-party proxy caches (Varnish or squid, for example) to extend `fs123`'s reach to thousands of client machines at multiple locations. Third, server upgrades (both hardware and software) need not interrupt client operation (just as HTTP servers are routinely upgraded on the Internet without client interruption). Fourth, HTTP supports the `stale-if-error` Cache-Control directive which will often allow `fs123` mount points to ride out temporary network outages with no user-visible consequences. In fact, a few users mount our software repository remotely from their laptops and work uninterrupted despite extremely spotty wireless coverage on their daily commute. In addition, one can take advantage of the rich ecosystem of HTTP capabilities, including TLS, URL-rewriters, logging, etc.

The `fs123` client and servers can optionally use a custom "secretbox" Content-Encoding (inspired by RFC 8188) that uses libsodium's shared key cryptography. With secretbox encoding, all data sent via HTTP is encrypted and authenticated in a cache-friendly way. Encoded data cached on disk and in public proxies, or in transit is unintelligible to anyone not in possession of a shared secret.

`fs123` is now a critical part of our infrastructure, in full production use for both programs and data, serving many billions of I/O requests per day. We have a software library of versioned executables, shared libraries, configuration files, etc., currently consisting of 1.1TB spread across approximately 22000 packages and 8M files. We also have in excess of 15 petabytes of simulation data, generated by our Anton supercomputers, and this figure is growing at roughly 10 terabytes per day. All of this (program and data) is accessed through `fs123`.

`fs123` is open source. We invite feedback and contributions from the community. Please see:

https://github.com/DEShawResearch/fs123