Fast and secure distributed read-only file system

Kevin Fu, M. Frans Kaashoek, and David Mazières

http://www.fs.net/

Submitted to OSDI2000

Introduction

- The Self-Certifying File System (SFS) provides a secure read-write network file system. However, there are some undesirable properties:
  - Bottlenecked by online public key cryptography.
  - Server's private key must reside online.
- Proposal: Develop a protocol specialized for public, read-only data.

Goals of the SFS Read-Only file system

- Integrity protection of public, read-only data
  - Protection of a *file system* rather than a *file*
- Scalability to many clients
  - Support many more clients than the SFS read-write server.
  - Precompute all server-side cryptography.
- Replication on *untrusted* servers
  - Servers hold no private keys.
  - Clients verify authenticity of server responses.

Background from network file systems

- Files are named by file handles, an opaque array of bytes.
- File systems store data in a hierarchical fashion: directories, indirect block pointers, etc.

What SFSRO borrows from SFS: Self-certifying pathnames

- **Make server keys explicit in file names**
- **File systems lie under `/sfs/`*Location*:*HostID***

$$\boldsymbol{HostID} = hash\,(PK_S, \boldsymbol{Location})$$

  - *Location* is DNS name or IP address
  - *HostID* is 20 bytes regardless of key length
  - Finding hash collisions considered intractable
- ***HostIDs* effectively equivalent to public keys**

Self-certifying pathname details

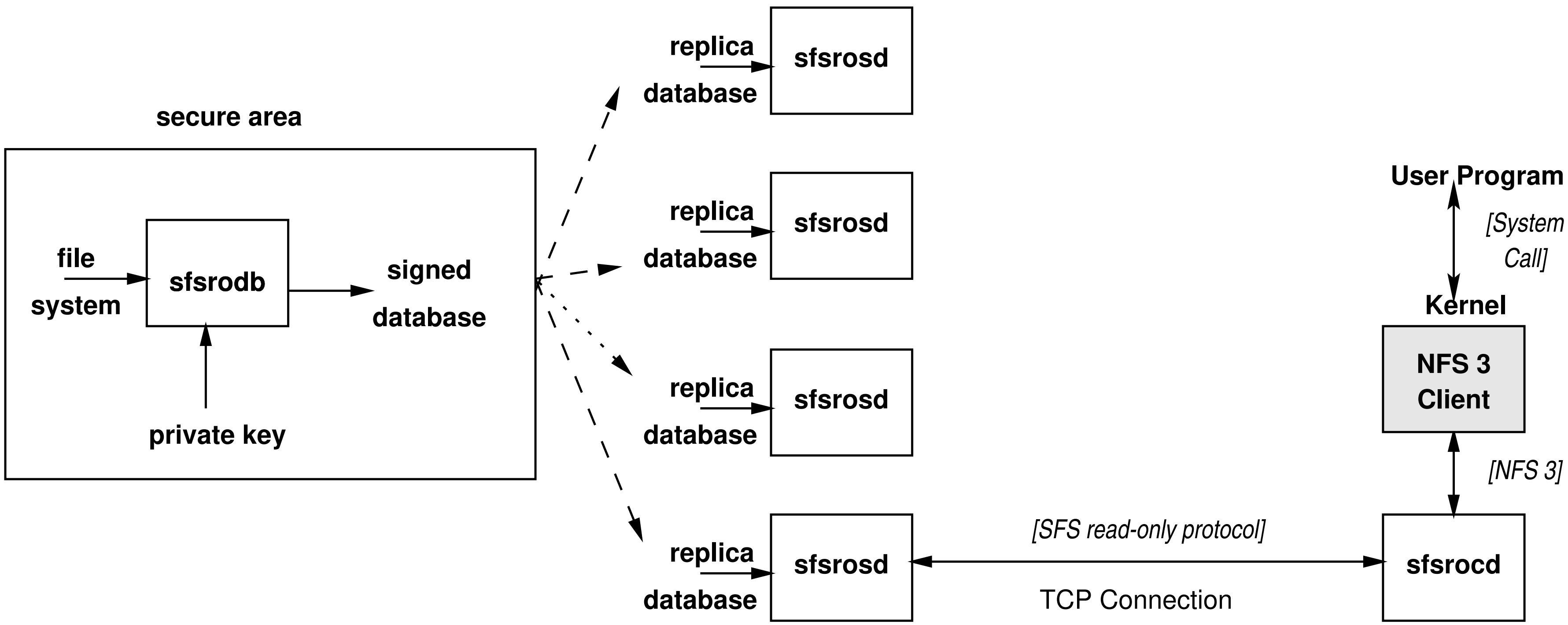
$$\texttt{/sfs/}\underbrace{\texttt{sfs.fs.net}}_{\textit{Location}}\texttt{:}\overbrace{\texttt{eu4cvv6wcnzscer98yn4qjpjnn9iv6pi}}^{\textit{HostID}\text{ (specifies public key)}}\texttt{/}\underbrace{\texttt{sfswww/index.html}}_{\text{path on remote server}}$$

- **Pathnames transparently created when referenced**
  - Anyone can create a server
  - New servers instantly accessible from any client
- **Client requires server to have *HostID*'s private key**
- **Pathname implies nothing about name of server**
  - e.g., server may not actually be the real `sfs.fs.net`
  - Need key management to produce the correct file name

Differences from traditional file systems

- Network file systems typically:
  - Run within the kernel.
  - Use file handles that contain the inode numbers from the local file system.
  - Do not address security.
- The SFS Read-only file system (SFSRO):
  - Runs at user-level, uses NFS 3 for portability.
  - Uses a cryptographic hash of data as the data's file handle.
  - Provides integrity protection and guarantees of recentness.

Architectural Overview

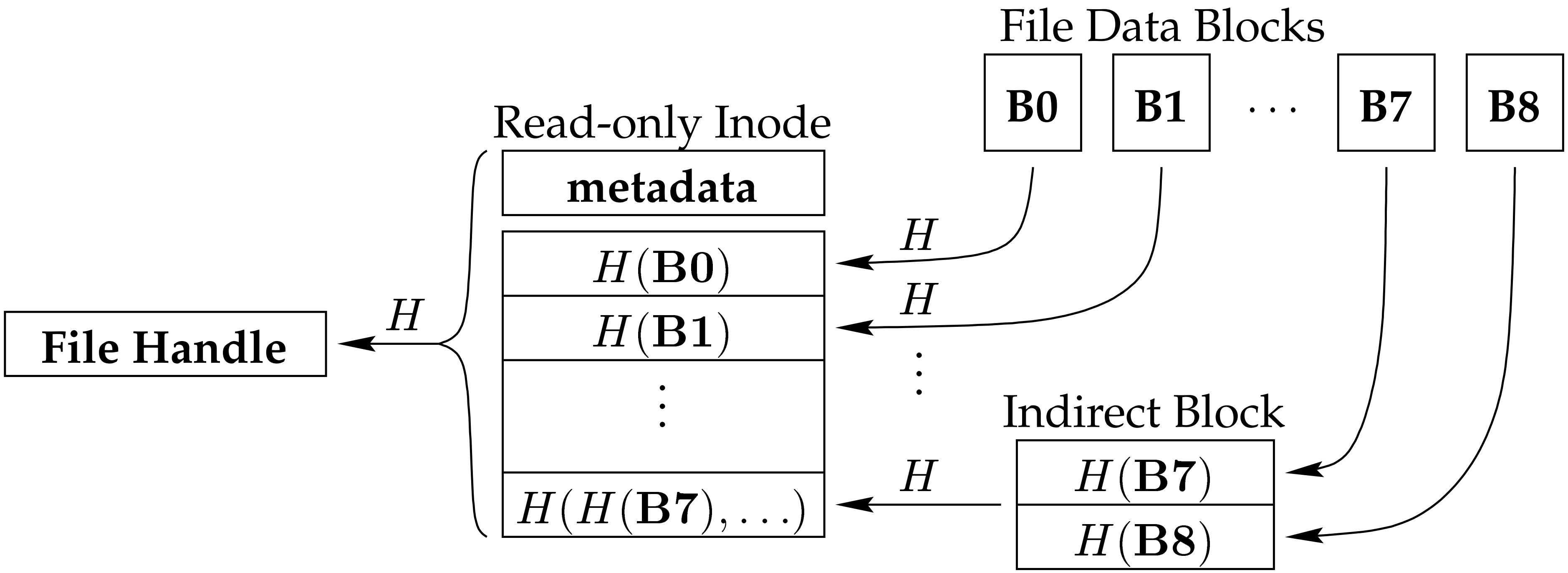


SFSRO Components

The SFSRO system consists of:

- Trees of cryptographic hashes
- Database generator (sfsrodb)
- SFSRO server daemon (sfsrosd)
- SFSRO client daemon (sfsrocd)

Tree of cryptographic hashes



All file system structures are named by cryptographic hashes rather than traditional index numbers.

Database generator (sfsrodb)

- An administrator runs sfsrodb in a secure area with access to the server's private key.
- Creates a hash tree of an existing file system.
- Stores marshaled data in the eXternal Data Representation (XDR) form
- After inserting a file system, adds a tree of of all file handles.
- Makes a single signature on the root file handle and other file system metadata.

SFSRO server daemon (sfsrosd)

- An extremely simple design ($<$ 400 lines of code).
- Responds to two Remote Procedure Calls: GETFSINFO and GETDATA.
- Follows in the footsteps of HTTP's simplicity (very few methods).
- Single threaded, asynchronous operation.

SFSRO client daemon (sfsrocd)

- Runs as NFS loopback server.
- Makes RPCs to sfsrosd and verifies integrity.
- Verifies the non-existence of file handles.
- Verifies the ".." relative directory.

Simplified example of a readlink

- **Read the file** `/sfs/sfs.mit.edu:bzcc5hder7cuc86kf6qswyx6yuemnw69/README`.
- **Client's local operating system contacts the sfsrocd NFS loopback server.**
- **sfsrocd calls GETFSINFO to retrieve the signed root file handle, verifies signature and start time.**
- **sfsrocd calls GETDATA to obtain the root directory's inode. Searches directory for** `README`.
- **sfsrocd calls GETDATA to retrieve the inode of** `README`.
- **sfsrocd calls GETDATA to retrieve file contents.**

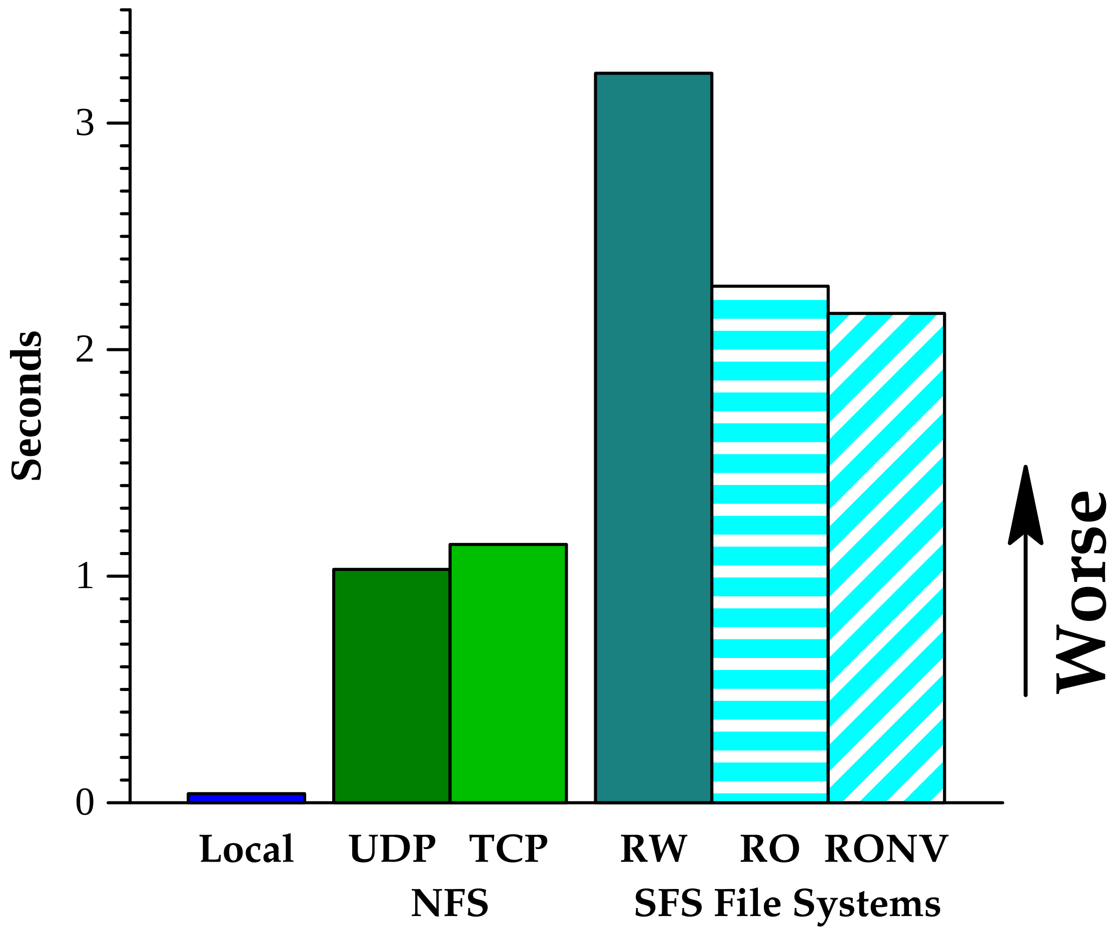
Applications

- **Certification Authorities (CAs)**
  - **Certify a server by creating a secure symlink.**
  - **SFSRO can better handle revocation.**
  - **SFSRO opaque directories for large listings.**
- **Software distribution**
  - **SFSRO allows mirroring of software distributions on untrusted servers.**
  - **Compliments signed RPMs by maintaining *distribution* rather than *individual RPM* security.**
  - **RPMs have no built-in revocation support.**

Performance Goals

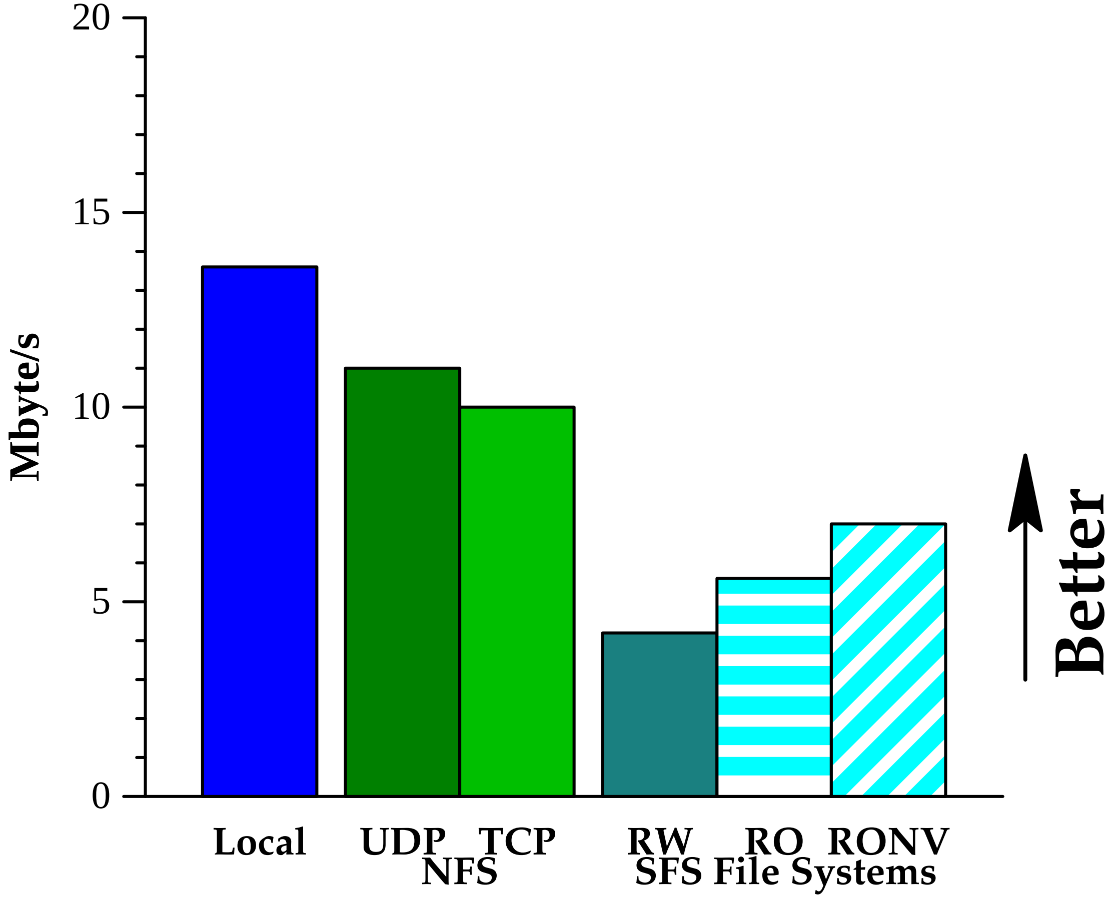
- Provide acceptable application performance.
- Scale well with the number of clients.
- Benchmarked on 550MHz Pentium III machines running FreeBSD.

Performance: Small file LFS benchmark



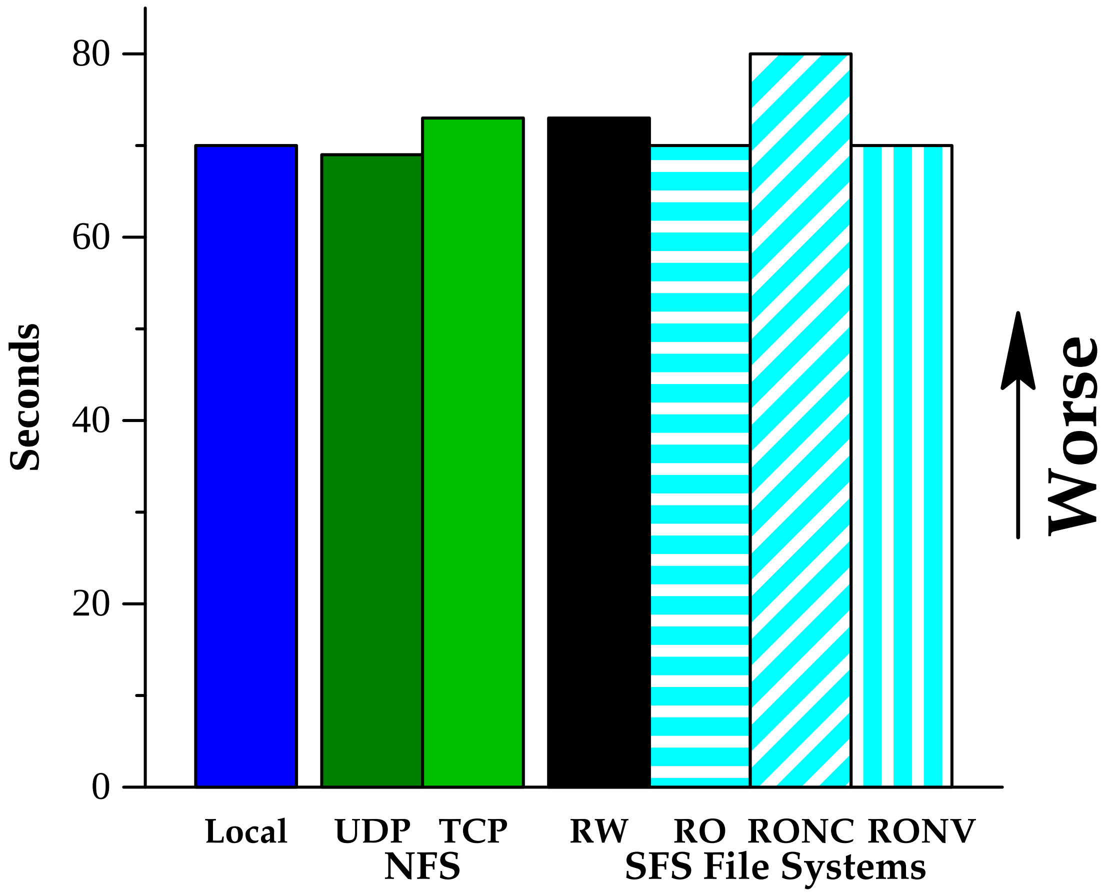
Time to read 1,000 1KB files. Shows that crypto cost is small relative to NFS user-level implementation and that SFSRO is faster than SFSRW.

Performance: Large file LFS benchmark



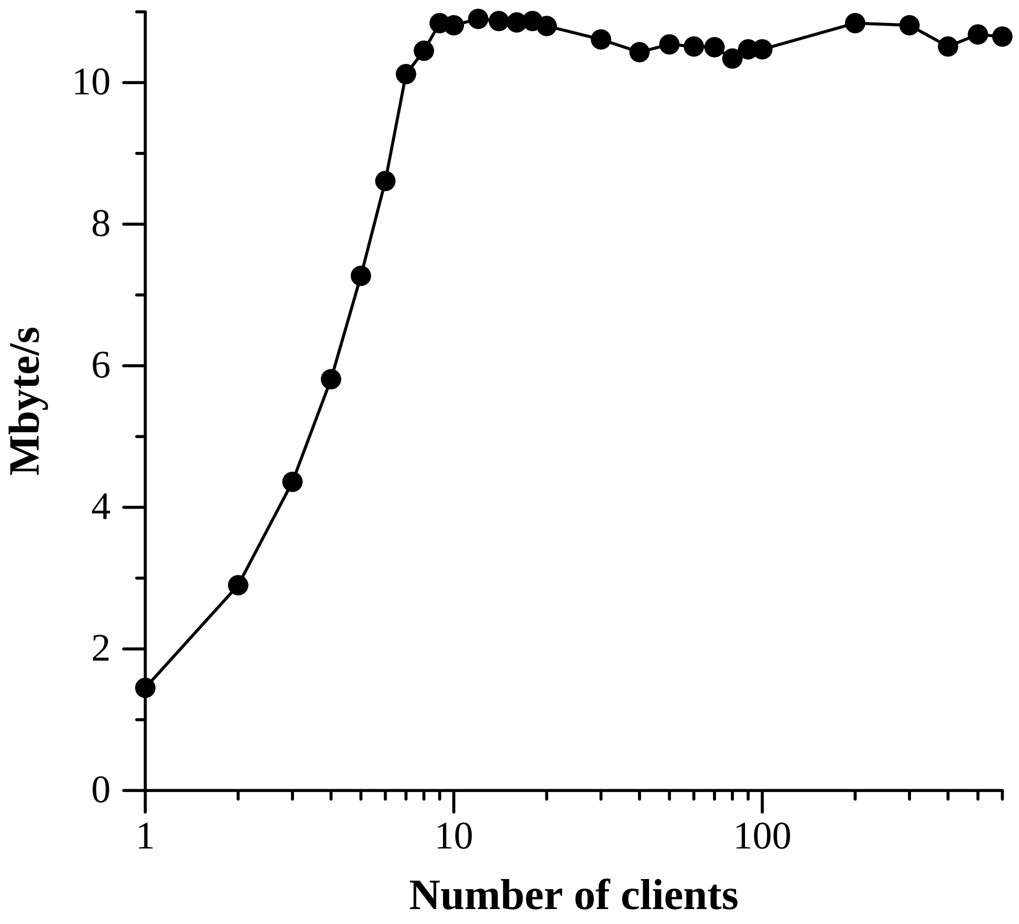
Throughput of sequential reads of a 40MB file. Shows verification costs only 1.4Mbytes/s.

Performance: Compiling Emacs



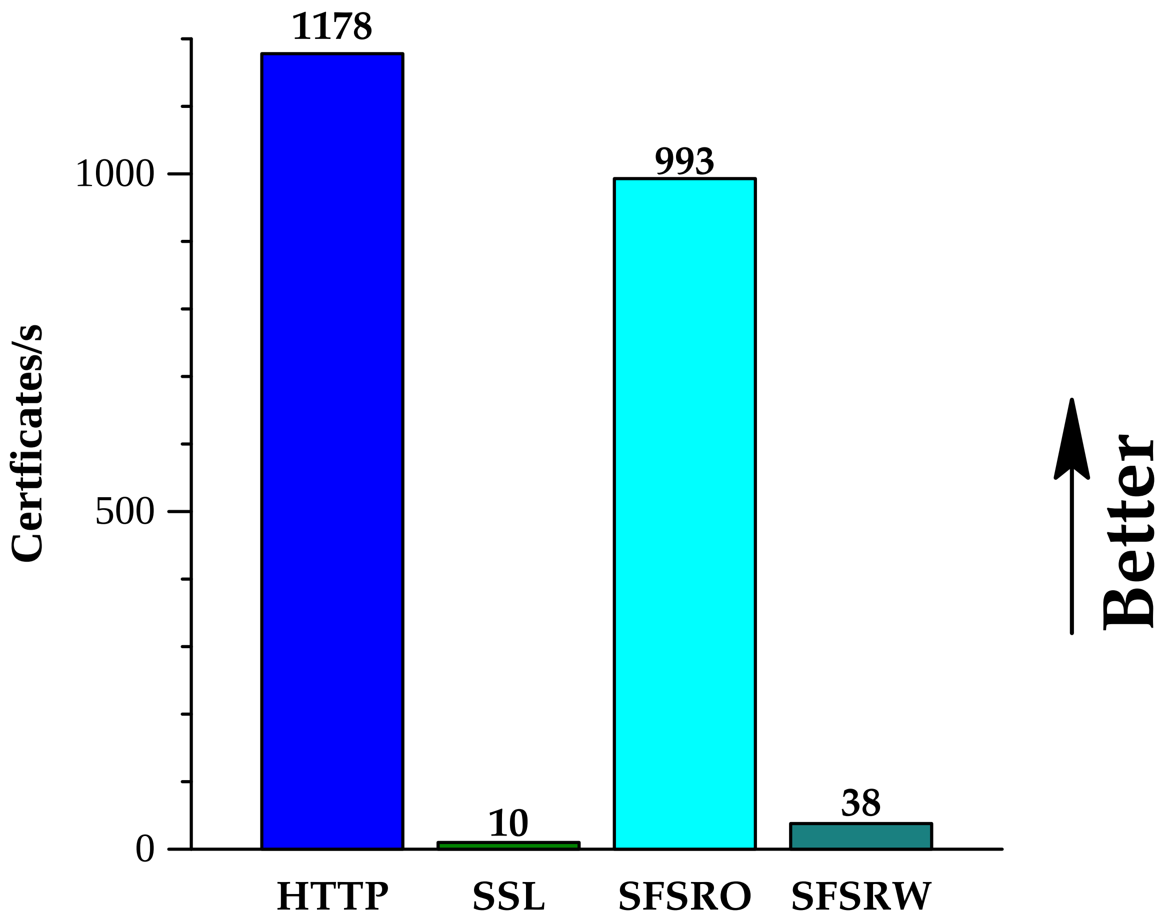
End user notices little or no degradation of application performance.

Performance: Number of clients



Throughput of server with increasing number of clients compiling emacs. We saturate the 100MBit ethernet at 10 clients.

Performance: Certificate Authority



For comparison, the root DNS servers handle about 4000 connections/sec.

Conclusions

- Distributed file system for a large number of clients.
- Integrity protection of public, read-only data.
- Signed offline.
- Able to replicate on untrusted servers
- Implementation shows we can support a large number of clients while maintaining acceptable application performance.

Download it!

SFS is free software.

`http://www.fs.net/`

or

`/sfs/sfs.fs.net:eu4cvv6wcnzscer98yn4qjpjnn9iv6pi/`