Caching in the Sprite network file system

Peter Couvares ("Scribe")
CS 739: Distributed Systems
University of Wisconsin, Madison

Friday, February 10, 2006
Spring 2006

1 Overview

This paper describes the use of caching in the Sprite network operating system, and compares the filesystem's semantics and performance to both tradtional local filesystems and other distributed filesystems such as AFS.

As in the earlier Sprite paper, the authors make a number of tradeoffs in their design between the goals of consistency, performance, scalability, and reliability. In general, they sacrifice the latter three for the first.

2 Problem Statement and Assumptions

Many of the Sprite Operating System's novel features, such as transparent process migration, rely on a distributed filesystem, where each client can read and write to a single, shared filesystem.

However, in a distributed filesystem without caching, the overhead of processing each file operation on two machines (client and server) and communicating it over the network can result in a significant performance disadvantage relative to local disk.

Likewise, in a distributed without consistency, applications must take care to coordinate in order to guarantee they don't operate on stale data.

The goal of the Sprite FS is to provide a single, uniform filesystem view to all the workstation clients, with performance comparable to (or better than) that of traditional local-disk filesystems. They wished to preserve as many of the traditional local-disk filesystem semantics as possible.

A key requirement was that any two clients should have exactly the same view of the global filesystem as they would if it was local to their own machine (in contrast to AFS).

This fits with the earlier Sprite paper's goals – remote operation should look exactly like local.

The Sprite FS was designed assuming a department-scale network consisting primarily of high-powered (potentially multi-CPU) diskless workstations supported by a central file service, all running the Sprite OS.

In retrospect, it seems a strange assumption that multiple CPUs would be common in a typical departmental workstation, but not disk. (One person commented that they seemed to have an "obsession" with diskless workstations.) Also, while their machines had a relatively small amount of memory compared to today, their workloads were equivalently small.

Accordingly, the workload was assumed to be that of a typical 1980's "Computer Science Department" user-base, primarily interactive software development. Therefore they designed for a workload with sequential file access, typically whole-file access, with many smaller files, and files open for a short time.

3 Methodology

The methodology of the paper was to design and implement a filesystem for the Sprite OS and measure its performance against other filesystems using a number of simulated workloads.

Although actual traces were used to inform the design, they were not used in the comparison of performance against other systems, which is a weakness of the paper.

4 Design and Implementation

The Sprite FS caches individual file blocks rather than whole files, and uses the system memory as the cache instead of the disk. *Block caching works well with memory, whereas file caching – as in AFS – works well with a disk cache.* Using system memory improves access speed, but limits the size of the cache. In order to allow their cache to use as much memory as possible without interfering with running applications, they allowed the cache size to dynamically grow or shrink. *Ultimately, the use of a dynamic cache may have been one of the more powerful*

ideas in the paper, as it has become a near-universal OS feature.

The Sprite FS was also implemented at the kernel level, as opposed to the user-level (like, e.g., AFS). *This improves performance at the cost of increased complexity and decreased reliability – as the Sprite developers themselves admitted in their earlier paper, when describing how often modifications to one area of their kernel causes problems in other places.*

As long as there are no writers, the Sprite FS performs read caching on all clients, in order to improve performance. This avoids both disk and network latency, and decreases load on the file server.

The Sprite FS performs write-back (rather than write-through) caching, unless there are other readers or writers. *The use of write-back caching acheives a dramatic performance improvement, but increases the liklihood of data loss in the event of a crash, when the data is not yet on disk.* Unlike AFS or NFS, which flush dirty files on close, Sprite flushes dirty blocks periodically (every 30 seconds). *This is somewhat alarming, as most writers assume their data is "safe" on disk after a close, whereas with the Sprite FS it is not.*

The Sprite FS allows sequential write sharing by forcing new writers' open calls to block until the previous writer's data has been flushed. It also allows concurrent write sharing by disabling caching altogether whenever there is a writer with any other readers of writers.

NFS does block caching in memory, and guarantees nothing for consistency except for flush on close. Its server is stateless, so clients are robust to server crashes. AFS also flushes on close; readers may have stale data (there are no updates of open files), but readers are guaranteed to see latest file on open. Sprite, by disabling write caching when there are any other readers or writers, ensures that all clients see the same files all the time.

Sprite's lack of callbacks simplifies implementation but means there is more server load on opens – but they claim their faster kernel-based networking made this less expensive than it was with AFS (which rejected the approach).

5 Conclusions

In the Sprite FS, performance, scalability, and reliability were clearly less important than consistency. Their cache is disabled in many instances in which AFS's is not (e.g., concurrent write sharing), hurting performance in many common scenarios. This may have been a mistake, as consistency is possible to ensure at a higher level, and scalability and performance may have limited the Sprite FS's adoption compared to AFS. Furthermore, as one student pointed out, the Sprite OS paper itself encourages the use of files for communication in Sprite – i.e., concurrent write sharing – which is a scenario in which their own FS would disable caching and perform poorly.