NOKIA

Research Center

NRC-TR-2008-004

File System Support for Low-Bandwidth Thumbnails

Jonathan Ledlie

Nokia Research Center Cambridge, US
http://research.nokia.com
May 6, 2008

Abstract:

Users are frustrated by the current experience of browsing, downloading, and exchanging files in low-bandwidth networks, such as ad hoc wireless networks. One improvement to this experience is to let users first locally interact with file thumbnails — graphical summaries of file contents — and fetch complete files only if necessary. These thumbnails can be sent quickly over the network and augment traditional metadata, such as file names and ownership.

To reduce bandwidth consumption, thumbnails must be generated remotely. In prior work, server-side application programs generate thumbnails in response to users' requests. In contrast, we propose extending underlying file system functionality to support direct generation and storage of thumbnails. Through inode changes or, more simply, through extended file attributes, we envision embedding thumbnails within the file system itself, just as human-readable names and ownership are today. Making thumbnails a first-class object directly linked to their uncompressed counterpart ensures consistency through modify-on-write regeneration and guarantees equal thumbnail and file data access control. Through directly embedding thumbnails within the file system, we can provide assurances and efficiencies beyond what is possible with application-level techniques.

Index Terms:

file systems
thumbnails
remote data access
file metadata

1 Problem Scenario

Consider a scenario where users want to collaborate and share their files through a wireless network. Each device (e.g., laptops, phones) might store different sets of files. Users want to be able to share any of their files and keep updates in sync in the future. However, due to bandwidth and storage constraints and due to periodic network partitions, it is neither physically nor theoretically possible to keep all files available and consistent.

Users need to be able to select which files they want to be actively shared and kept in sync. For example, they might browse their friend's device and want to select certain files and directories, activating them for download and future syncing. A similar situation occurs when a mobile user wants to access his or her own files that are stored primarily on a non-mobile desktop or server. Current file system interfaces offer the user two choices: (1) fetch the file metadata only, or (2) fetch both the file metadata and the file itself.

Because file metadata only includes information such as the name, size, and modification date, this is generally not sufficient for users to know if they are interested in files (think of video or image files named *345436566.jpg*). Thus option (1) is insufficient.

Option (2) — fetching the full file and then generating a thumbnail — is impractical: files can be arbitrarily large and fetching full file contents makes little sense without a strong certainty the user needs local access to the whole file.

In addition to active browsing, users may also want to browse "offline" — when their friend's set of files is not actively available. Again, traditional metadata, even if it is cached on the local device, will generally be inadequate for offline use. In particular, names, ownership, access times, etc. do not provide sufficient information about a file to help the user know if it should be actively synchronized in the future.

While these examples refers to a mobile user, they could equally apply to any low-bandwidth environment such as a rural network in a developing region.

2 Solution

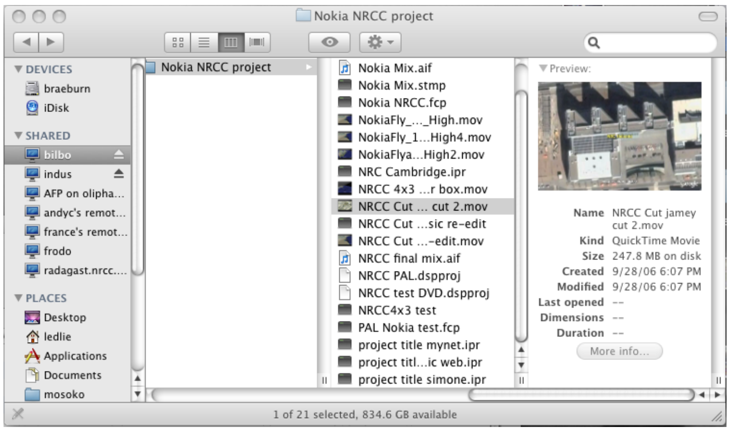
Instead of fetching whole files, we propose the remote file system itself generates and stores application-specific thumbnails of each file along with traditional metadata. We envision thumbnails as typically being graphical, compressed representations of the full file contents, but textual or other representations are possible. This metadata plus compressed data summary eliminates the problem of saturating the network with unwanted files (option (2) above), while providing the user with much more information than simple traditional metadata (option (1) above).

3 Prior Work

Using Apple's Finder [1] over a remote file system may appear similar (particularly if one is using a high speed link), but is actually option (2) above. Instead of explicitly sending a thumbnail preview of each file, Apple's Finder fetches the full file and then locally generates a thumbnail (for more information see Apple's Quick Look Programming Guide). This technique is not feasible on storage and bandwidth constrained devices.

Figure 1 presents a screen shot of Apple's Finder summarizing a remote file. The summary includes a thumbnail image of the file. The thumbnail was generated locally, after the file was fetched. Instead, we propose that the thumbnails be generated and stored remotely, but fetched along with the file metadata. This provides the benefit of a thumbnail without the cost of whole file download.

Copernic's Mobile Search [2] provides an application-level solution to the problem of efficiently browsing remote files in a low-bandwidth environment. In contrast to Apple's Finder, Copernic uses a server-side component installed on the user's primary machine to generate thumbnails in response to client requests. We compare this approach to ours in Section 5.



Nokia NRCC project

DEVICES
braeburn
iDisk
SHARED
bilbo
indus
AFP on olipha...
andyc's remot...
france's remot...
frodo
radagast.nrcc....
PLACES
Desktop
ledlie
Applications
Documents
mosoko

Nokia NRCC project

Nokia Mix.aif
Nokia Mix.stmp
Nokia NRCC.fcp
NokiaFly_..._High.mov
NokiaFly_1...High4.mov
NokiaFlya...High2.mov
NRC Cambridge.ipr
NRCC 4x3 ...r box.mov
NRCC Cut ... cut 2.mov
NRCC Cut ...sic re-edit
NRCC Cut ...-edit.mov
NRCC final mix.aif
NRCC PAL.dspproj
NRCC test DVD.dspproj
NRCC4x3 test
PAL Nokia test.fcp
project title mynet.ipr
project titl...ic web.ipr
project title simone.ipr

Preview:

Name	NRCC Cut jamey cut 2.mov
Kind	QuickTime Movie
Size	247.8 MB on disk
Created	9/28/06 6:07 PM
Modified	9/28/06 6:07 PM
Last opened	--
Dimensions	--
Duration	--

More info...

1 of 21 selected, 834.6 GB available

Figure 1: Apple's Finder generates previews of files, including videos, images, and documents. To create these thumbnails when files are stored remotely, it transfers the entire file and creates the preview on the local device. In low-bandwidth environments, such as inter-mobile device exchange and server-to-device download, this method can swamp the network.

4 Implementation

Two orthogonal issues are central to implementing file system support for thumbnails: (1) storage location and (2) triggering thumbnail (re-)creation.

Most major network file protocols and file systems (*e.g.,* NFS, NTFS, ext2, ext3, HFS+, WAFL) already support *extended attributes*: data associated with a file, but not interpreted by the file system or operating system. Thus, using this extended attribute mechanism is the most straightforward place to store the thumbnail, and is preferable to inode modification.

There exist several options for triggering thumbnail creation and re-creation: (1) on file save or update (*i.e.,* modify-on-write), (2) on remote file fetch, or (3) in the background via a daemon. (1) and (2) have the disadvantage that, if implemented poorly, they could affect the user's experience — the user would wait while the thumbnail was generated. (3) has the severe disadvantage that it is nondeterministic. Because thumbnail generation will be quick with the file open and already in memory and because thumbnails will typically not change even when the underlying file changes (*e.g.,* they will show only the first frame of a video), updating the thumbnail on file save is arguably the best approach.

The operating system is aware of writes to each file and can trigger thumbnail generation on each such event. Similar to the association of mime-types to viewers, we propose each mime-type have an associated thumbnail generator. These generators can be installed and altered by an administrative user. In order to limit spurious thumbnail re-generation, generators can keep state associated with each file. The implementation for these generators already exists in what are now client-side thumbnail generators.

This model of extended attribute storage and action-on-write is inspired by the work of Muniswamy-Reddy *et al.,* on Provenance-Aware Storage Systems [3]. In this work, provenance, *i.e.,* the complete history of a file, is stored in extended attributes and changes are triggered on writes to the file. For example, the provenance of a compiled object file includes references to the specific compiler and all input files. Tracking provenance is significantly more complex than thumbnail generation because provenance must transfer across machines (including those without extended attributes) and because file histories can be inter-dependent. Their work, however, demonstrates that triggered actions on write can effectively save newly-generated information into extended attributes without diminishing the user's experience.

5 Advantages and Disadvantages

Apple's approach to client-side thumbnail generation has certain advantages. The primary advantage is that it is agnostic to the remote file system type. That is, no changes are required for the server data store in terms of application software or file system internals. The primary disadvantage, as noted above, is bandwidth consumption. Because much or all of each files' contents must be sent over the network in order to generate a thumbnail, locally-generated thumbnails are not appropriate when clients receive bandwidth is limited, when the sender's bandwidth is limited, or when a large number of clients are sharing the wireless medium. The same issues occur in limited or oversubscribed wired networks.

Copernic's solution shares advantages and disadvantages with other designs that choose to function above the operating system interface. Its main advantage is that it requires no change to the file system and can immediately function on top of an existing one. This design parallels work on application-level provenance trackers (*e.g.,* [4]). These systems maintain file provenance in distinct database systems, tracking changes through application-level triggers. As with provenance data, this introduces problems such as: "ensuring consistency between provenance and the data, enforcing provenance maintenance, and preserving provenance during backup, restoration, and copying" [3]. In addition, security and access controls to the two sets of data must be kept in sync. As with other attributes that are available through the file system, incorporating thumbnails directly makes them available to *all* applications through a common and well-understood interface. As is typically the case with attributes that would benefit a broad class of applications, embedding the service directly into the operating system — in this case, the file system — outweighs the short-term cost of operating system modification.

References

[1] Apple finder. `http://www.apple.com/macosx/features/finder.html`.

[2] Copernic mobile search. `http://www.copernic.com`.

[3] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-Aware Storage Systems. In *Proc. of USENIX Annual Technical Conference*, Boston, MA, June 2006.

[4] G. Singh, S. Bharathi, A. L. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. In *Proc. of High Performance Networking and Computing (Supercomputing)*, Phoenix, AZ, Nov. 2003.