Poster at EuroSys 2008, Glasgow, UK

Hades-Managing storage in caching proxies for distributed filesystems

Lamprini Konsta and Stergios V.Anastasiadis
Department of Computer Science, University of Ioannina, Greece

Introduction

- Recent trends in business and research collaboration within and across organizations encourage secure data sharing over wide area networks
  - minimal intervention of the end user
  - best possible performance
- Traditional file transfer mechanisms such as FTP have long been used for secure data and file transferring
  - Get the user explicitly initiate the whole file transfer
  - Manage multiple copies
- Caching proxies have been lately introduced as an alternative approach
  - Reduce WAN latency by caching data closer to the client
- Nache: a representative example of a caching file server proxy for NFSv4
  - Designed to retain a consistent cache of remote file servers in a distributed environment
  - Improves file accesses performance by redirecting requests that were initially intended for the file server to the intermediate cache proxy

Research

- Topic
  - Caching proxy facilities for distributed filesystems
- Focus
  - Efficient storage management
  - Improve the performance of accessing cached data from proxies
  - Make performance comparable to or better than direct disk accesses from local file system

Background

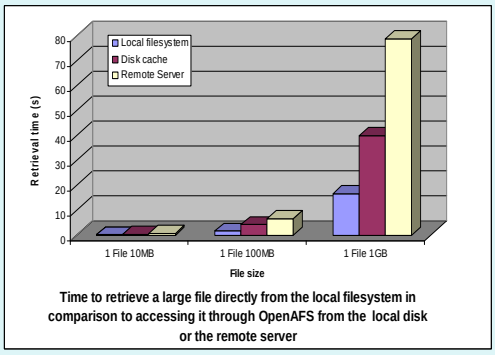
- Traditional distributed file systems
  - Originally designed for serving the storage needs of users at local area networks
  - Limited client-side caching to main memory or local file system
  - Made unnecessary the corresponding disk-based caching
- Network-based filesystems, like Andrew File System
  - WANs introduce latencies that may be orders of magnitudes greater than direct disk accesses
  - Long latencies encourage the design of a client-side disk cache for effective storage management

Andrew File System

- Distributed file system that is better used for file-sharing in wide-area networks
  - Provides the capability to temporarily store data in a disk cache at the local file system of each client machine
  - Improves scalability and availability in distributed environments
  - Reduces network traffic and server load
- Chunk file
  - data transfer and store unit
- It initially creates a number of fixed-size chunk files in the local filesystem of the client
  - Used to store chunks of remote data
- Successfully used in general file systems
  - Does not offer a proxy caching service
  - Limits caching to the local filesystem
- Distributed environments in engineering and scientific collaboration
  - Transfer numerous small or enormously large files
  - Existing approach of one local file per chunk file is not the best possible solution in terms of data access or metadata management efficiency

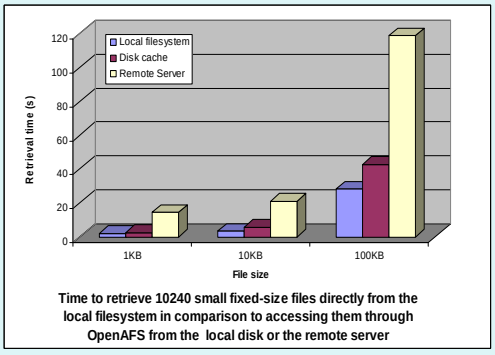
Experiments

- Platform
  - Open-source variant (OpenAFS v.1.4.5) over Linux kernel v. 2.6.18
- Parameters
  - Default chunk size of 256 KB
  - Cache partition in the local filesystem large enough to store the requested files in their entirety
- Measurements
  - Local file access time
  - Disk cache file access time through OpenAFS
- **Results**
  - One large file
    - The time to fetch a large file from the remote file server is **1.5-2 times greater** than the corresponding retrieval time from the OpenAFS disk cache
    - The time to retrieve a large file from the disk cache through OpenAFS is **2-3 times greater** in comparison to retrieving it from the local file system



Time to retrieve a large file directly from the local filesystem in comparison to accessing it through OpenAFS from the local disk or the remote server

  - Numerous small files
    - The time to fetch numerous small files from the remote file server is **2.5-5.5 times greater** than the corresponding retrieval time from the OpenAFS disk cache
    - The time to retrieve numerous small files from the disk cache through OpenAFS is **1.3-1.8 times greater** in comparison to retrieving them from the local file system



Time to retrieve 10240 small fixed-size files directly from the local filesystem in comparison to accessing them through OpenAFS from the local disk or the remote server

Measurements

- One large file
  - Stored in multiple chunk files
  - Large time to find and read all the chunk files for a single remote file
  - Open/close overhead
  - Metadata management overhead
  - Fragmentation/Disk access overhead
- Numerous small files
  - Open/close overhead
  - Metadata management overhead
  - Fragmentation/Disk access overhead

Approach

- Explore alternative methods for the mapping of the remote data to local files in disk cache to
  - enhance the existing performance of retrieving cached data
  - match or improve the performance of accessing data directly from the local file system
- Thus, we propose two methods that change
  - the existing mapping strategies
  - the way remote data chunks are organized into local cache files

Existing methods

- One chunk file per remote file
  - The dominant organization in recent published literature
  - Offers a consistent view of the remote data as they appear at the remote server
- Multiple chunk files per remote file
  - AFS mapping strategy
  - Depending on the size of the remote file, it is saved in one or more files in local disk cache
  - Appropriate structures are used to find all the corresponding cached files for a remote file
- Multiple remote files per chunk file
  - Web proxies
  - Manage local data in a way that serves their design objectives

Proposed methods

- Storage management in cache
  - separate from storage management at the remote server
- Remote small files
  - Organize in multiple files per local chunk file
  - Create few large chunk files in cache
  - Store the remote chunks of data on demand
- Remote large files
  - Organize in one or few local chunk files per remote file
  - Contiguously store chunks of the remote file to the appropriate local chunk file on demand

Expected results

- Improve performance by reducing
  - storage space fragmentation
  - metadata management overhead

References

[1] A.Gulati et al., *Nache: Design and Implementation of a Caching Proxy for NFSV4*, USENIX FAST, Feb 2007
[2] M. Satyanarayanan, *Scalable, Secure and Highly Available Distributed File Access*, IEEE Computer 23(5): 9-21 (1990)
[3] E. Markatos et al., *Secondary Storage Management for Web Proxies*, USENIX Symposium on Internet Technologies and Systems, 1999
[4] Gopalan Sivathanu and Erez Zadok, *A Versatile Persistent Caching Framework for File Systems*, Stony Brook University, Technical Report FSL-05-05, 2005

For further information

- Please contact *{lkonsta, stergios}@cs.uoi.gr*
- More information on this and related projects can be obtained at www.srg.cs.uoi.gr
- Supported in part by Interreg IIIA Greece-Italy 2000-2006 Grant No I2101005 in the framework of the project "Interstore: Decentralized data sharing with applications to biomedical image processing"

Hellenic Republic · European Commission · Italian Republic
INTERREG IIIA GREECE-ITALY PROGRAMMING PERIOD 2000-2006

Hades – Managing storage in caching proxies for distributed filesystems[1]

Lamprini Konsta*
Stergios V. Anastasiadis
Department of Computer Science
University of Ioannina, GREECE
{lkonsta, stergios}@cs.uoi.gr

Current trends in business and research collaboration encourage secure data sharing over wide-area networks with minimal intervention of the end user. Instead of having users explicitly initiate traditional file transfers that replicate datasets close to computation resources, it would be preferable to have a caching proxy to automatically replicate datasets and hide transfer delays during the repetitive use of data. Lately, the design of caching proxies for distributed filesystems is attracting research interest from the systems community mostly in terms of getting existing filesystems interoperational with local file systems for persistent caching purposes. Here we point out the need for efficient storage management in caching proxies so that accesses of cached data from the local disk of the proxy have performance comparable to or better than direct disk accesses from the local file system.

Caching proxies behind web servers have already been broadly used for over a decade in content distribution networks. Originally, copies of web pages requested by users were replicated on proxy servers close to the web browsers over traditional local file systems (e.g. UFS). However, related experimentation in published literature demonstrated several performance deficiencies related to metadata management of multiple small files, frequent creation and deletion of files, excessive disk head movement from poor clustering of jointly used data or access overheads from multiple small writes. Subsequently, customized file systems emerged that complementarily addressed the above issues through special internal architectures and new access interfaces.

On the other hand, most distributed filesystems were originally designed for serving the storage needs of users within the same organization at a single geographical site. The assumed use of a local-area network limited client-side caching to main memory and made unnecessary the corresponding disk-based caching. One notable exception is the Andrew File System and its descendants that provided the capability to temporarily store data at the local file system of the client machine for purposes of improved scalability and availability in distributed environments [1]. However, Andrew makes the basic assumption that client machines belonging to individual users are powerful enough to relieve centralized servers from computations which might not be necessarily the case when building caching proxies to be shared by large numbers of clients within an organization.

In fact, Andrew replicates remote data in chunks of a configurable fixed size. Initially, it creates a large number of individual files on the local file system of the client and subsequently uses each of them to store an individual chunk requested from the server. In the case of general file system use, Andrew has been widely successful for over two decades. However, in modern scientific and business environments it is usual to have extreme cases of numerous small files or enormously large ones. Then, the approach of having a separate local file per chunk might not be the best possible in terms of data access or metadata management efficiency.

The dominant organization in recent published literature is to map each remote file to a local file in the caching proxy [2]. However, we claim that apart from offering a consistent view of the remote data as they appear at the remote server, the caching proxy should be free to manage its local data in whatever way serves its design objectives better. The experience from the web proxy research [3] may provide useful lessons with respect to avoiding unnecessary metadata management or improving disk access efficiency through appropriate file structuring approaches or data layouts on disk.

In our effort to understand the caching efficiency of Andrew, we experimented with its open-source variant (OpenAFS v. 1.4.5) over Linux kernel v. 2.6.18. We assume the default chunk size of 256KB and sufficient disk space to fit the entire requested dataset. As we see in Filgure 1, the retrieval time from the disk cache through OpenAFS is about 2-3 times greater for large files and 1.3-1.8 times greater for numerous small files in comparison to the retrieval time from the local file system. On the other hand, fetching files from the remote server (another node on the same gigabit Ethernet switch in our experiments) costs about 150-200% the retrieval time from the OpenAFS disk cache for large files and 250-550% for numerous small files.

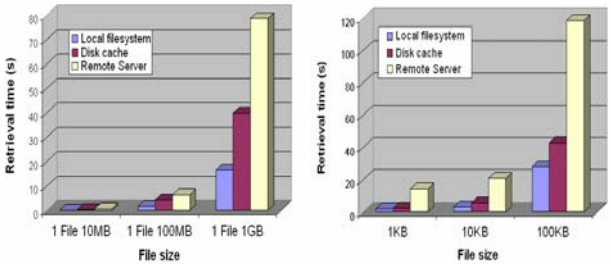


Figure 1. Time to retrieve one large file or numerous small files directly from the local file system in comparison to accessing them through OpenAFS from the local disk or the remote server.

In our current research, we use kernel profiling to explain the above overhead of OpenAFS. In general, we plan to use OpenAFS as a testbed for experimentally evaluating alternative policies of mapping remote data to local files and organizing data chunks within each local file in a way that matches or improves the performance of accessing data directly from the local file system.

[1] M. Satyanarayanan, *Scalable, Secure, and Highly Available Distributed File Access*, IEEE Computer 23(5): 9-21 (1990).
[2] A. Gulati et al. *Nache: Design and Implementation of a Caching Proxy for NFSv4*, USENIX FAST, Feb 2007.
[3] E. Markatos et al., *Secondary Storage Management for Web Proxies*, USENIX Symposium on Internet Technologies and Systems, 1999

* Graduate student

[1] Supported in part by INTERREG IIIA Greece-Italy Grant No I2101005.