Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

1

Introduction

1.1 What is a network storage service?

A network storage service is a long term data repository in a Distributed Computing Environment (DCE) through which users may collaborate and share information resources.

Typically, users gain access to a DCE through workstations and share resources, such as computation servers, network gateways, etc, on the network. Similarly, the task of providing long term data storage for all the storage needs in a DCE is better met by a network storage service than by scattered and localised workstation storage. There are several reasons. Firstly, for economic reasons, there is often a strong incentive to share resources. Secondly, as the data stored on computers are a vital asset to users and form the primary input and product of information processing, it is of great importance that the security, integrity and availability of the data be maintained and the data be retrieved efficiently. Data security can only be enforced if the data are completely under the control of a trusted authority. Data integrity requires protective measures, such as routine backup, and to be administered to guard against storage media failure and user errors. Data availability demands speedy recovery from system faults or data replication to mask out system failures. The demand for speedy data access can best be met by exploiting the latest advances in hardware and software technologies through purpose built servers. All these criteria favour a network storage service under centralised administration.

Distributed file systems are today's state-of-the-art network storage service. A distributed file system stores files which are named objects that are created and destroyed by the system in response to explicit external commands. The files remain in existence and are immune to temporary system failures until their explicit destruction.

An important characteristic of (distributed) file systems is that they do not know the type and the logical relationships of stored data items. This is the most important difference between a (distributed) file system and a (distributed) database. Data items in the latter can be accessed associatively according to some user specified predicates.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

1.2 Research Motivation

This work observes that contemporary distributed file systems are optimised to store text and binary data. Structured and continuous-medium data are becoming important data types. Digital audio and video are examples of continuous-medium data now being deployed in computing environments. These data types are very different from conventional text and binary and cannot be supported by distributed file systems.

Furthermore, contemporary distributed file systems are designed in a highly vertical integrated fashion and do not provide a proper framework for extensions. Extensibility is necessary both in enhancing the storage service functionalities and for the utilisation of new storage device types and data placement/migration strategies.

1.3 Research Statement

This work proposes a new network storage service design– the Multi-Service Storage Architecture (MSSA).

MSSA conforms to the file storage model, i.e. objects are stored and retrieved by names. However, MSSA is different from contemporary distributed file systems because it supports multiple file abstractions. Contemporary distributed file systems only support the UNIX-style flat (or byte-stream) file abstraction. MSSA supports others, such as the structured and continuous-medium file abstractions, as well.

MSSA is also structurally different from contemporary distributed file systems. Modularity and Extensibility are the main emphases of this work. Common functionalities are factored into layers. Each layer adds value to the services provided by the set of lower layers. This multi-layer approach to support new data types also distinguishes MSSA from other work on new data type support, especially in the field of continuous-medium data storage.

1.4 Dissertation Outline

Chapter 2 reviews the development of distributed file systems and points to the limitations of these systems. The chapter also discusses several emerging trends which will have significant impact on the design of storage services. The discussion concludes that a new approach to storage service design is necessary.

Chapter 3 summarises the previous observations into four goals that MSSA should achieve. This is followed by a presentation of the MSSA framework. MSSA internally is a two-layer architecture which supports different file abstractions. The chapter also defines the interface between the two layers and introduces the notion of sessions which is a new way to specify performance requirements dynamically.

Chapter 4 and 5 address several issues that must be tackled with common policies in all file abstractions. These include access control, naming, object location, and existence control.

Chapter 6 presents the design of a byte segment custode (BSC) which is a lower layer component of MSSA. The BSC provides atomic update semantics with a transaction mechanism. The transaction mechanism uses non-volatile memory to achieve high perfor

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

mance. Its performance is evaluated in chapter 7.

Chapter 8 discusses the concept and interface of *rate-based* sessions. *Rate-based* sessions allow the upper layer to obtain temporal performance guarantee from the lower layer in order to support the real time delivery of continuous-medium data.

Chapter 9 describes a prototype implementation of *rate-based* sessions and presents an evaluation of its performance.

Chapter 10 concludes this dissertation with a summary of the lessons learned and some suggestions for future work.

© in this web service Cambridge University Press www.cambridge.org

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

2

Background

2.1 Introduction

The purpose of this chapter is to establish the context for discussion in the rest of this dissertation. This work is motivated by the recognition that contemporary distributed file systems have basic assumptions that limit their applicability to newly emerged data types and applications. To understand these limitations, it is important to look at the development of distributed file systems in the past 15 years (section 2.2) and the design tradeoffs that have been made (section 2.3).

The progress in computer technologies has brought about new applications which were not possible only a few years ago. However, these applications also brought about new problems that must be solved. In sections 2.4,2.5,2.6 and 2.7, four emerging trends that have significant impact on the design of network storage services are discussed. This chapter concludes with an assertion that a holistic design restructuring is necessary.

2.2 The Development of Distributed File Systems

The earliest model of file sharing, dating back to the early 70s, involved user-initiated file transfers between two machines on the computer network. Users were fully aware of the distinction between local and remote files, both in naming convention and permitted operations. Today, this approach is no longer regarded as a distributed file system. Nevertheless, file transfer programs such as FTP [Bhu71] and FTAM [fta85] are still used to share files, especially over national or global networks.

The ability to perform the same set of operations on both local and remote files was soon recognised as an important property of distributed file systems. In fact, this property, commonly known as network transparency, has since become a fundamental requirement which affects many aspects of distributed file system design.

In the early 80s, there was considerable interest in providing **atomic transactions** and **concurrency control** in distributed file systems. Transaction support was considered a

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

useful mechanism to prevent inconsistencies arising from machine and communication failures or concurrent access to files by other clients. Felix [FO81], XDFS [SMI80], Alpine [BKT85], Swallow [Svo81] are examples of such systems. Since then, the UNIX-style byte stream file model [Bac86] [MJLF84] [POS90] has been adopted by most distributed file systems. None of these provides direct transaction support. As the systems are mainly used for program development and engineering applications, this is considered a reasonable tradeoff between high performance and the possible danger of having occasional data inconsistency.

The Cambridge File Server (CFS) [Dio80], developed in the early 80s, differed from other designs in the same period in one significant respect. The Cambridge design is a capability-based virtual disc system augmented with a naming substrate which provides the desired coherency without committing the clients to any specific directory structures or textual name conventions. The CFS was used to support two file systems (Tripos [RN83] and the CAP filing system [Del80]), an object-based file store [Cra86] and a digital voice store [Cal87].

Since the mid-80s, researchers have explored the use of **data caching** on diskless or dataless[1] workstations to improve the performance of remote file serving. The cache consistency problem is addressed in different ways depending on the models of data sharing adopted by the different designs. The Cedar File System [SGN85], which was the first caching file system, eliminates the cache consistency problem by forcing all files to be immutable. Sun's Network File System (NFS) [Sun89] does not define clearly what guarantees the system makes about the consistency of the client caches. In most NFS implementations, data are used directly from the cache if it has been validated within some time period, usually a few seconds. Andrew [HKM+88], MFS [Bur88] and Sprite [NWO88], developed around the same time, adopt different write-sharing models and assume a different granularity of sharing. Hence the cache consistency protocols of the three designs are quite different. A comparison of the relative merits of the various cache consistency protocols is contained in [Bur88].

The **availability** of distributed file systems has been a concern of researchers for many years. LOCUS [WPE+83] is an early example which combined replication and an algorithm to detect inconsistency to achieve high availability. In the past few years, there has been considerable interest in using file replication to achieve high availabi lity. Echo [HBM+89], Coda [SKK+90], Harp [LGG+91] and Deceit [SBM89] are examples of highly available distributed file systems. The systems differ from earlier designs in that they are all Unix-style file systems and have combined client caching with server replication.

Disconnected operation forms a new strand of work in distributed file system research. This line of work recognises the importance of mobile computing and the need to support automatic data consolidation when a portable computer is reconnected to a distributed computing environment (DCE). Coda [KS91] is the first system to support this mode of computing.

Distributed file system has been an active area of research for nearly two decades. It is not possible to cover, in this limited discussion, all the significant work. More detailed accounts can be found in [Svo84] [LS90]. However, it is clear from the discussion above

[1] This refers to workstations which have local disks set up as virtual memory paging store and/or temporary file systems only.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

that distributed file systems have been designed to fulfill four requirements: efficiency, transparency, reliability and security.

In other words, a distributed file system should be as *efficient* to use as local file systems and yet its distributed nature should be *transparent* to users. As a shared resource, its capacity and performance have to be *scalable* to match the increase in the size of the system. Being a central point for data sharing, it has to be *reliable* and highly *available*. Finally, it has to be adequately *secure* to prevent unauthorised access to valuable data stored in the system.

However, like other engineering work, the design of distributed file systems always involves tradeoffs between desirable system behaviour and the cost and limitation of hardware and software technologies.

2.3 Limitations of Today's Distributed File Systems

In distributed file system research, many well tried ideas are highly dependent on the *physical* and *usage* characteristics of data. The size distribution of files, the access pattern, the update pattern, the degree of concurrent access, the granularity of concurrent access, the availability requirement and the level of tolerance towards loss of data in events of system failures, etc., are all important design considerations.

For example, client caching plays a very important role in the success of distributed file systems. The implicit assumption is that the client access pattern exhibits good locality of reference and client caches are large enough to achieve a high hit rate. Also, the cache consistency protocols assume low levels of concurrent write-sharing. For instance, the early version of AFS [HKM+88] only supports whole file caching and sequential write sharing because these were thought to be the common characteristics of data sharing in a DCE. Although the derivative of AFS— OSF DEcorum [KLA+91]— and other caching file systems [Bur88] [NWO88] support strict single-system semantics, these systems are likely to operate efficiently only if the degree of concurrent sharing is low.

Some research work on improving the performance of file systems relies heavily on the observed access patterns to make the design tradeoffs. The Bullet file server [vRTW89] is one example. The designers observed that most files in the file system they had studied were small in size and were usually accessed in their entirety. To take advantage of larger main memory and higher capacity disks and to optimise for performance, the bullet file server only supports immutable files which are stored contiguously on disk. Files are cached in their entirety in RAM on the server. When compared with the SUN NFS using simple tests [vRST88], the designers claim that their server performs favourably and is 3 to 6 times faster on reads. However the design has a major potential drawback in that its functionality and performance would be degraded far more severely than more conventional file system designs, such as the Unix FFS [MJLF84], when the data properties are not what the designers expect. For instance, the strict immutability of files requires new copies be created for every update and the extra overhead of copying old data would become intolerable for frequent, and perhaps small step, updates.

Similarly, log-structured file systems, such as Sprite LFS [RO91], are designed with the assumption that large client caches will have the effect of shifting the server work load towards being write-dominated. The basic principle of this work is to collect large amounts of new data in a file cache in main memory and then write the data to disk in a single large

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

disk write that can use all of the disk's bandwidth. However, this idea is complicated by the need to maintain large free areas on disk. Rosenblum reports that a simple cleaning policy based on cost and benefit analysis works well with some simulated and real work loads.

The issue of interest in this dissertation is not whether these techniques are suitable for today's research and engineering environments where the main data types are text and binaries and the principal activities are software development, engineering simulations and text processing. Rather, the assertion here is that new data types, which will be discussed in section 2.4, are very different from conventional data types in terms of *physical* and *usage* characteristics. Therefore, contemporary distributed file systems, being limited by their basic design assumptions, are unable to support the new data types.

Also, file servers are designed in a highly vertical integrated fashion and there is little room for extension. It would be difficult to extend such a design to accommodate new data types or other functional enhancements. Apart from the need to support new data types, there are other incentives to move towards a more extensible design. The incentives identified in this dissertation are: better support for structured data (section 2.5), extensions to primary storage functions (section 2.6) and better data placement strategies (section 2.7).

2.4 Support of Continuous-Medium Data

Digital video and audio are the latest members of information media being used in a computing environment. They are called continuous-media because, unlike other information types, there is an inherent temporal relationship between the sequence of discrete data samples of which they are composed.

A continuous-medium sequence is transmitted from one end point to another in real-time as a stream of data samples. The sequence has an inherent temporal function which determines when each data sample should be presented at the receiving end. Unfortunately, the data path between the two end points can introduce delay and delay variations (referred to as jitter). Therefore, it is necessary to control the magnitude of jitter to ensure an acceptable quality of presentation. Moreover, a multi-media presentation can, by definition, consist of more than one continuous-medium sequence. In that case, not only are the data samples within one sequence temporally related, but the data samples of different sequences must also be so.

As for the storage of continuous-medium data, the main source of jitter comes from the access time of rotating disk technologies. The sustainable data rate of a magnetic disk is limited as much by the seek and rotational latency as by the platter data transfer rate. More important, the variation in access time depends on the layout of data on disk. In conventional file systems, access time variation is never a deciding factor of data placement. Therefore, it is unlikely that the access time variation is bounded to a level suitable for supporting continuous-media. The problem is compounded by the need to support multiple presentations at the same time. The interference from other presentations, if uncontrolled, can exacerbate the access time variation.

The size of continuous-medium data is another problem. For instance, at 44.1kHz sampling rate and 16 bits per sample, compact disc (CD) quality digital stereo sound consumes roughly 10 Mbytes of storage per minute (171 Kbytes per second). The figure for video is even more staggering. For instance, when digitised at NTSC TV quality, i.e. 512x480 pix-

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

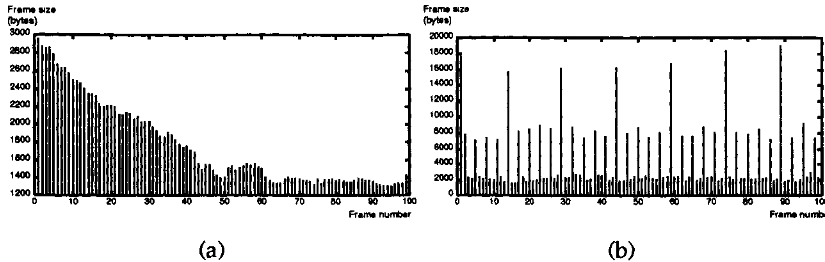

Figure 2.1: Sample frame-size-profiles of MPEG. The figures show the size of consecutive frames in two MPEG video sequences. Figure (a) is the profile of a computer graphics animation. The sequence only contains I-frames which are the compressed images of complete frames. The frame size varies with the complexity of the picture. Figure (b) is the profile of a scenery shot. The sequence contains I-frames interleaved with B- and P-frames. As B- and P-frames are the delta-encoding of I-frames, they are much smaller in size. Hence, the profile is more bursty than (a).

els, 8 bits per pixel for colour and hue and 30 frames per second, the data rate is over 7 Mbytes per second. HDTV quality video would require a much higher data rate. Clearly, continuous-medium data have to be compressed before transmission or storage on disks. Compressed digital video is by nature bursty (figure 2.1). With delta-encoding, which is used in most video compression standards, there is typically a large spike followed by smaller frames of motion changes. The allocation of (disk) bandwidth to match the bursty nature of compressed video is a problem.

Editing of continuous-medium data may need extra storage service support. For instance, voice data are stored as *ropes* in the voice storage server of the Etherphone System [TS87]. Each rope is implemented internally as lists of pointers to immutable voice recordings. The server supports editing operations, such as cut and paste, on *ropes* by manipulating the pointers. The purpose is to minimise the amount of copying involved in editing. It is true that, by today's standard, the size of main memory available is often enough to store an entire voice message and editing can be done entirely in memory like ordinary text data. However, video and HiFi audio can still be too voluminous to handle entirely in main memory and it can be too time consuming to copy during editing. Therefore, a level of indirection between the media data and the user-level abstraction, like the *rope*, seems an appropriate way to minimise copying.

The proliferation of digital video standards presents another problem to continuous-medium data storage. As Liebhold [LH91] points out, there are a number of digital video standards, some for full-bandwidth video and some for compressed representations (CD-I [Inc89], DVI [Lut91], JPEG [Wal91] [CH91], MPEG [Gal91], px64 [Lio91]). The variety of formats means that the timing information necessary for the timely delivery of data is encoded in different ways. It is a challenge to design a storage service that can cope with this diversity.

The topic of continuous-media is an active area of research. The nature of continuous-media presents a challenge to different disciplines. This work focusses on the impact of

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

continuous-media on the design of a storage service. The problems raised in this section regarding the storage of continuous-medium data will be addressed in a later part of this dissertation.

2.5 Support of Structured Data

Structured data are objects which consist of component objects. The components may be structured themselves and may be stored separately and referenced from the parent object by name.

A multi-media document is an example of structured data. It is a composition of different information media, such as text, graphics, audio and video. Each component may be stored as a distinct entity and referenced by a "shell" object. Also, the information regarding the presentation order of the various media components has to be stored separately and is likely to be structured as well.

A list of records, whether each record is fixed or variable length, is another example of structured data. A lot of information in real life is record oriented. Not surprisingly, much of the data stored as conventional unstructured files, for instance electronic mail folders, are in fact lists of variable length records. Similarly, the persistent data of database programming systems are structured as well.

A structured object has to be converted from its volatile representation in memory into a suitable form for storage and vice versa. The storage representation should contain sufficient information so that all component objects can be retrieved and reconstructed in memory. The conversion between memory and storage representation can be quite involved if the object structure is complex. If the size of a component is variable and becomes too large to fit into its existing slot, extra space has to be allocated for it. This may not be an easy task especially when the size of the structured object makes it very expensive to overwrite the object entirely when a component is changed.

The Unix file system does not support structured data. Other file systems, such as IBM's ISAM (Indexed Sequential Access Method), may have the notion of records (and a key searching capability). However they do not support variable size records or embedded references to other files. This leaves the applications to invent their own schemes to store structured data.

A question relevant to this work is whether a storage service should support structured data directly. There are several potential advantages of direct support:

1. The structure of the data is itself information that would otherwise be hard to represent.

2. The chance of applications misinterpreting the structure of a piece of data is reduced.

3. The storage service may take advantage of knowledge about the structure of data to improve performance.

4. The storage service can keep track of object references inside structured objects, and detect and remove unreachable objects.

Sue Thomson's work [Tho90] on structured data storage is directly related to this dissertation. She proposes a High Level Storage Service (HLSS) to store structured data. HLSS

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-55115-1 - A Modular and Extensible Network Storage Architecture
Sai-Lai Lo
Excerpt
More information

has two primitive types (byte sequence and object identifier) and a small set of constructors (sequence, record and union). The intention is to support primitive types that correspond to the smallest intended unit of storage access. The task of HLSS is to provide efficient storage access while leaving the fine granularity sub-object organisation to the applications.

The detailed design of structured data storage is not a primary concern in this work. However, the architectural framework proposed in this dissertation incorporates structured data storage as an integral part of the storage service.

2.6 Extensions to the Primary Storage Function

The primary function of MSSA is to store named objects persistently. This section discusses two examples of enhancing this primary function by value-adding clients.

2.6.1 File Indexing

As the volume of data stored in a file system increases, it becomes increasingly difficult to locate the data item one wants. In relation to this problem, some researchers [Sal91] [Sat91] have suggested that it will be very useful to extend the file system by adding associative search or indexing functions. An interesting piece of work in this area is the semantic file system by Gifford et al. [GJSJ91], which provides associative access to attributes extracted from the contents of files.

Many tools exist for associative search and indexing. A simple example is the UNIX command "grep" which searches for the occurrence of some regular expression in some files. However, the semantic file system is special in several aspects. Firstly, it provides automatic extraction of attributes from files with file type specific *transducers*. *Transducers* are programs that understand the content of files and are capable of extracting *useful* attributes from the files. Secondly, the *transducers* work behind the normal file interface and react to changes to files. Therefore, applications and users can access files as before and automatically obtain the additional benefit of associative access to the attributes collected by the transducers. Finally, it overloads the directory tree with the query facility. A query is composed into a directory path name and the result is obtained by listing the directory. The researchers claim that overloading the directory tree with query semantics can improve system uniformity and utility.

The semantic file system has been built from a Unix file system overlaid by a file server process (figure 2.2). The file server process exports an NFS interface. File content attribute queries, in the guise of directory look-ups, are trapped by the file server process. Normal file traffic goes through unchanged to the Unix file system. However, the file server process records file modification events in a write-behind log. An indexing process examines this log and responds to a modification event by re-indexing a file with the appropriate transducer.

Automatic extraction of attributes from files and the associative access to these attributes is an area which deserves more thorough investigation than can be afforded in the time and scope of this work. However, this area of work illustrates the need to extend a storage service by *adding value* to the core function of the service which is to provide long term data storage. The semantic file system work points to the idea of extracting file

© in this web service Cambridge University Press www.cambridge.org