Cegor: An Adaptive Distributed File System for Heterogeneous Network Environments

Weisong Shi, Sharun Santhosh, and Hanping Lufei
Wayne State University
{*weisong, sharun, hlufei*}*@wayne.edu*

Abstract

Distributed file systems have been extensively studied in the past, but they are still far from wide acceptance over heterogeneous network environments. Most traditional network file systems target the tight-couple high-speed networks only, and do not work well in the wide-area setting. Several communication optimization techniques are proposed in the context of wide-area file systems, but these approaches do not take into consideration the file characteristics and may instead introduce extra computing overhead when the network condition is good. We envision that the capability of providing adaptive, seamless file access to personal documents across diverse network connections plays an important role in the success of future distributed file systems.

*In this paper, we propose to build an adaptive distributed file system which provides the "***C***los***E** *and* **G***o,* **O***pen and* **R***esume" (Cegor) semantics across heterogeneous network connections, ranging from high-bandwidth local area network to low-bandwidth dial-up connection. Our approach relies on a set of new techniques for managing adaptive access to remote files, including three components:* system support for secure, transparent reconnection *at different places,* semantic-view based caching *to reduce communication frequencies in the system, and* type-specific communication optimization *to minimize the bandwidth requirement of synchronizations between clients and servers.*

1. Introduction

We have witnessed unprecedented growth in the number of mobile users, mobile devices, and connection technologies. This trend is enabling the vision of a pervasive computing paradigm where users have network access anytime, anywhere. For example, a well-traveled laptop user might use half a dozen different networks throughout the course of a day: a cable modem from home, a wide-area wireless on the commute, a wired Ethernet at the office, a Bluetooth network in the car, and a wireless, local-area network at the airport or the neighborhood coffee shop. It is now substantially easier to access public information from anywhere with the emergence of the World Wide Web (WWW). However, secure remote access to personal files and data after reconnection at other places still remains painful and cumbersome, which consists of three steps: (1) retrieve the files using secure file transfer tool, (2) work on the local file, and (3) write it back to remote file server using secure file transfer mechanism. Although we can access remote files by using telnet or some remote desktop tools [5], the poor network performance is quite frustrating and inconvenient.

Network and distributed file systems have been extensively studied in the past, but they are still far from wide deployment across diverse network connection technologies. Most traditional network file systems target the tight-couple high-speed network only, and do not work well in the wide-area setting [2, 3, 10]. Several communication optimization techniques are proposed in the context of wide-area file systems [13, 15, 23], but these approaches do not take into consideration the file characteristics and may instead introduce extra computing overhead when the network condition is good. The objective of this paper is to build an adaptive distributed file system which provides transparent "**C**los**E** and **G**o, **O**pen and **R**esume" (Cegor) semantics across heterogeneous network environments, ranging from high-bandwidth local area network to low-bandwidth dial-up connection.

There are three challenging issues in building an adaptive distributed file system: (1) *trustness* The new environment to which the user has connected is untrusted in general, and so does the remote client to the file server; (2) *diversity* Diverse network connection technologies on which the distributed file systems operate demand that **adaptation** should be the first-class option during the design; (3) *transparency* A transparent reconnection mechanism which provides "close and go, open and resume" experience [8] is of great importance to the wide acceptance of a distributed file system. However, these issues receive little assistance from today's operating systems or Internet protocols.

Our approach relies on a set of new techniques for managing adaptive access to remote files. This includes three components: *system support for secure, transparent reconnection* at different places, *semantic-view based caching* to reduce communication frequencies in the system, and *type-specific communication optimization* to minimize the bandwidth requirement of synchronizations between clients and servers. Specifically, the contributions of the Cegor file system include:

- *A transparent reconnection mechanism*, which provides the client the same connection view as the last connection. View credentials are used to authenticate remote clients.
- *An adaptive lease-based reconcile consistency* to accommodate heterogeneous network environments. The number of reconciliations between clients and servers are adapted based on current client connection technology.
- *A semantic view based zooming protocol* which efficiently maintains the consistency between multiple intermittent clients and the file server. The adaptive zooming protocol has the potential to adjust communication frequencies of clients.
- *A novel memory cache replacement policy* that uses the observed semantic distance between files to improve replacement decisions.
- *An adaptive type-specific communication optimization scheme* to minimize communication bandwidth requirements of messages without introducing any extra network round-trips.

The above techniques will be incorporated into a prototype implementation called Cegor based on OpenAFS [25], which is an open source implementation of the AFS system [10]. Note that the Cegor system is built targeting for a small collaborative group of co-workers, typically in academic research group or business project group. However, it is also designed to scale for large number of such groups.

The rest of the paper is organized as follows. Section 2 depicts the detailed design of Cegor, including three major components: secure, transparent reconnection, semantic-view based caching, and type-specific communication optimization. Related work is discussed in Section 3. Finally, current status and future work are presented in Section 4.

2. Design of Cegor File System

We first briefly describe an overview of our approach. Given this summary, the rest of this section presents authentication and transparent reconnection, semantic view based caching, adaptive type-specific communication optimization and some preliminary results.

2.1. Overview of Our Approach

Our objective is to build an adaptive file system which provides the "**C**los**E** and **G**o, **O**pen and **R**esume" (Cegor) semantics across diverse network connections. Our approach to build the Cegor file system consists of three components: *secure and transparent reconnection*, *semantic view based caching*, and *adaptive type-specific communication optimization*.

Our approach for secure and transparent reconnection for mobile users is based on the notion of connection view and a home-based authentication protocol. The connection view of a client is a snapshot of current connections between the client and the file server (Section 2.2.1). After reconnecting from another place, Cegor will automatically rebuild all existing connections between the client and the server based on credentials embedded in connection view (Section 2.2.2), and maintain consistency based on semantic views (Section 2.3.1).

We view the problem of performance optimization in a distributed file system as two sub-problems: *reducing the number of messages (bandwidth saving)* and *reducing bandwidth consumption of update messages (latency reduction)*. Our approach for the first sub-problem is based on client-side file caching and a relaxed consistency model. The core idea behind our approach is the notion of "semantic view", which depicts the current file view of client cache (Section 2.3.1). We propose a semantic view based reconcile consistency protocol (Section 2.3.2). Based on this model, two optimizations, *shadow cache and adaptive zooming protocol* (Section 2.3.3) and *semantic-distance based cache replacement* (Section 2.3.4) are proposed to further reduce the number of messages.

We address the second sub-problem by proposing a *type-specific differencing* technique. Our preliminary results show that different file types have different characteristics, which motivate us to apply different optimization techniques against them. To adapt to different connections and diverse computing overhead of different differencing algorithms, we propose a dynamic adaptive scheme to automatically negotiate the communication protocol between clients and servers (Section 2.4).

2.2. Secure and Transparent Reconnection

The major security concern of an adaptive distributed file system with nomadic support is *client request authentication and authorization*, which addresses the problem of the distribution and enforcement of access rights to clients. This is especially important when nomadic clients are connected from other administrative domains, such as airports and hotels.

2.2.1. Client Request Authentication and Authorization In NFS, a server needs to add an entry in its file export configuration file (e.g., `/etc/exports`) to allow a directory accessible by other machines (based on their IP addresses). Later, the server controls accesses to this directory based on this file. In a nomadic setting, in which a client usually gets an IP address from a new environment

dynamically, this IP address based exporting mechanism is inflexible for transparent access. Moreover, the userid based authentication over a wide-area network is vulnerable. Therefore, an IP-independent, userid-independent authentication and authorization approach is required to support transparent remote access. Although the self-certified file name approach shares the same goals as ours [20], our approach are completely different.

We introduce the concept of *connection view (CV)* to support secure access and transparent reconnection over heterogeneous network environments. Logically, a connection view is the snapshot of all connections between a client and a file server when disconnecting. Each connection view has a view credential which includes the access key of this client's view root (e.g., the mount point of this client), current working path(s) of related processes running on the client, file read-write rights, and related file-read keys and file-write keys. The connection view credential will be used for reconnection authentication later.

The file server (actually file owner) controls read-write rights of files (including directories) by assigning file-access keys [12]. Logically, each file has two keys: *file-read key* and *file-write key*. Since it is impractical to have two distinct keys for each file within one connection view, we propose a general approach for key derivations from the root key of a view, as described in the following.

Consider a connection view CV_i, and let the file-read key and file-write key of the root be K_{ro} and K_{rw}. Let m be some node (file or directory) within the connection view, and let $name(m)$ return the name of node m, and $parent(m)$ return the node id of m's up directory. The file access keys for node m, K_{ro_m} and K_{rw_m}, are derived as follows:

$$K_{r(o|w)_m} = \begin{cases} K_{r(o|w)} & : \text{ if m is root} \\ H(name(m)|K_{parent(m)}) & : \text{ otherwise} \end{cases}$$

where, $H(.)$ is a one way hash function, which is computationally infeasible to invert. We intend to use SHA-1 [1] for this purpose. Note that, the keys of all files and directories within the connection view can be derived from the root key by default. However, in some circumstances, the server may want to restrict the access rights of some files or directories from some users. In this case, it is easy for the server to assign a separate sub-root key for these files or directories, and the keys associated with children of the directories will be derived in the same way as before. File-access keys distribution is accomplished by introducing another key named as the *key-lock key*. The key-lock key is a pre-determined key for this file server, which can be distributed to legal file clients by any out-of-band security means. During the initial mounting procedure, the file-access keys of the root directory will be piggybacked to the client, encrypted by the shared key-lock key.

It is easy to authenticate client requests given the key derivation algorithm. Let us assume that a secure channel has been constructed between a (remote) client and a file server no matter where the client being, any request for a filesystem operation will be verified using the appropriate request credential (RC). To prevent the replay attacks, a request credential includes three parts: a random generated nonce n, a timestamp t, and $H(K_{r(o|w)}|n|t)$, where $K_{r(o|w)}$ is the appropriate file access key. This form of credential prevents the eavesdropping of the key and assures the server that the requester does possess the proper key.

Cegor provides an adaptive authentication policy to address the diverse security requirements. For example, in a local area network which supports Kerbores [24], the access control can be implemented by Kerbores. However, when the client connects to a different Kerbores domain, a connection view based authentication protocol will be used, as discussed in the following.

2.2.2. Transparent Reconnection It is straightforward to support secure and transparent reconnection given the notion of the connection view and the view credential. Before the disconnection, the client creates a reconnection credential by hashing the value of his connection view, and sends this credential back to the server. Later, this connection credential is used by the HAP protocol to authenticate the requested client. After that, the client part of the Cegor system will automatically mount the filesystem and set the current working path(s), and create multiple shell windows (if necessary) at the client side. To improve the performance, modifications made by other clients during this period are proactively send to the client side by Cegor.

2.3. Semantic View Based Caching

The central idea of Cegor is using adaptive caching techniques to amortize the heterogeneity of network technologies. We borrow the idea of *whole-file caching* from AFS [15], but we provide another layer of abstraction — *semantic view*— to make the system adaptive.

2.3.1. Semantic View The semantic view of a client is defined as the snapshot of all files and directories stored in the client cache, including names and attributes (also known as metadata). Semantic views provide Cegor with the essential ability to maintain consistency between client and server and to optimize communication.

Logically, a semantic view includes three parts: *files and their attributes*, *directories and their attributes*, and *semantic distance of these files and directories*. Each file is associated with a six-element tuple:
$< fileID, op, rw, V_{last}, \mathcal{D}, \mathcal{N} >$, where `fileID` is the unique file identifier in the Cegor system; `op` is used to describe the current status of the file, with its three possible values *access*, *cache* and *shadowed* (the discuss of shadowed cache is detained to next section); `rw` represents the access control to this file by this client; Versioning is used in Cegor to maintain consistency and reduce communication overhead, V_{last} is the last version number at

the end of this session. A session is defined as the interval between two synchronization points between the client and the server; $\mathcal{D}$ is the digest of file content, calculated by SHA-1 function [1]; $\mathcal{N}$ is the set of files whose semantic distance to this file is less than a predefined distance threshold ($D_{threshold}$).

Cegor treats directories in the same way as files, however, the closest neighbor set of a directory is defined as the set of directories whose directory semantic distance to this directory is less than a predefined value. The notion of semantic distance between two files is defined as the lifetime semantic distance, i.e., the overlap time of the accesses to any two files, borrowed from original work of SEER [16]. We extend this notion to support directory semantic distance, which is defined as the average file semantic distance between all files within two directories: $D_{AB} = \frac{1}{N_A \times N_B} \sum_{i=1}^{N_A} \sum_{j=1}^{N_B} d_{ij}$, where N_A, N_B are the number of files in each directory. If there is no semantic relationship between two files, the semantic distance is defined as the maximum distance D_{max}.

2.3.2. Reconcile Consistency Protocol To support "always on" connectivity, the consistency protocol of Cegor includes three parts: (1) normal operation; (2) stop operation; and (3) resume operation. We will describe them separately in this section.

Normal operation During the normal operation between clients and servers, Cegor adopts an adaptive lease based approach. The lease time of each client is negotiated at the beginning of connection setup and determined by the current connection speed. During the course of two renewals, all file operations except an open file missing are performed locally. When the lease time expires, the client needs to send back his semantic views to the server, together with all the file and directory changes made during the last session. To save the communication overhead, the last semantic view of each client is maintained at both sides so that only the difference between the new semantic view and the last semantic view needs to be sent across the network. After the server gets the new semantic view from the client, it will check the current status of all files and directories, and send back the modifications that were made by other clients. We propose a novel consistency model called *reconcile consistency* in Cegor for maintaining coherence between multiple clients and servers.

The reconcile consistency is motivated by the following example: assume there are two files, A and B, shared by two clients, C_1 and C_2. Assume C_1 caches A, and C_2 caches both A and B. Suppose that C_2 updates A and B, and then reconciles with the server. The server now knows about both new versions. If C_1 wants to reference both A and B without an intervening reconciliation among C_2's reconciliation, it will see the old version A (cached) and new version of B. Therefore, client C_1 sees updates out-of-order. We believe that it is very important for the Cegor system to provide consistent in-order updates for each client. Continuing with the previous example, when client C_1 wants to access B, the server should send back an old version that should be visible to C_1 after the last reconciliation. Therefore, we have the following definition:

Reconcile Consistency: *A client needs to reconcile with the file server when its lease is expired. Between two reconciliation events, all the local cache missing in the client will be served with the corresponding latest version, which chronologically happened before its last reconciliation. It is the reconcile consistency that guarantees this client will see the new version of all files by the next reconciliation.*

Reconcile consistency assures that the modifications made by one client A are guaranteed to be visible to another client B within at most two rounds of reconciliation — one is between A and the server, the other is between B and the server. In other words, the maximum delay between a write operation from one client to a read operation of another client is between 0 and $(T_i + T_j)$, where T_i and T_j are the current lease time of client i and j respectively. We believe that this is an acceptable delay in a distributed system. To reduce the communication overhead further, the server will not send back client i's recent modifications of those files whose operating mode is in *access* status at client i. Reconcile consistency eliminates the requirement to maintain read and write token holders at the server side for providing tight consistency, used by Sprite [26] and Echo [9].

Cegor supports multiple-readers/single-writer semantics based on a modified optimistic concurrency control strategy. Similar to the conventional Unix file system, exclusive file locks are used to prevent multiple writers concurrently modifying a file. However, if a user has the read-write right (in its last semantic view) and wants to modify this file in the middle of a reconciliation session, Cegor allows the user to optimistically start modifying before obtaining the exclusive file lock,thus deferring the file lock acquisition to the next synchronization point. Although there are certain risks of modification conflicts, we believe this is rarely happens based on previous conflict analysis on AFS [15] and Ficus [27].

Stop operation Because both the server and the client have no idea of how long it will take before the next reconnection, it is better for the client to send back all the modifications that he has made since the last synchronization and release all file locks it has occupied. Cegor forces a synchronization between the disconnecting client and the server even though the lease remains effective. All files in access operating mode at the client need to be synchronized, and all exclusive file locks should be released.

Resume operation After successfully reconnecting based on the mechanisms described in Section 2.2.2, Cegor negotiates a new specific lease time T_{lease} for this new connection based on the estimated bandwidth and latency be-

tween client and server. Also, the client-side of Cegor needs to synchronize with the server to get the last version of all files in its local cache based on this client's last semantic view.

2.3.3. Shadow Cache and Adaptive Zooming Protocol During the procedure of reconciliation in the basic Cegor consistency protocol, the modifications made by other clients must be synchronized to satisfy the reconcile consistency requirement. However, this is not adaptive enough for heterogeneous network environments. In this subsection, we propose an adaptive zooming protocol based on a new concept of shadow cache to optimize the adaptability of Cegor.

The basic idea behind shadow cache is motivated by the fact that if a local cached file is changed frequently by other clients but not accessed by a local client for sometime, it is better to remove it from local cache. On the other hand, it is better to keep it in the memory cache to reduce the future communication bandwidth. For example, if this file is accessed again, we can fetch the difference between this version and the latest version from the server instead of fetching the whole file. To solve this conflicting requirement, we propose to split a traditional memory cache into two parts: *regular* and *shadow*. The shadow cache is used to keep those files that are evicted because of temporary infrequent access. Later, when these files in the shadow cache are accessed again, it needs to be revalidated with the server immediately before it is returned .

Based on the idea of shadow cache, we propose two adaptive zooming approaches to exchange files between the regular cache and the shadow cache. When a client has a low bandwidth connection, a multiple-decrease (zoom out) and single-increase (zoom in) of its semantic views will be adopted during the view zooming procedure. Single-increase means that only the accessed file itself will be added from the shadow cache to the regular cache (shadowed files are accessed again), while multiple-decrease means that multiple related files plus the selected file will be evicted from the regular cache. On the contrary, a single-decrease (zoom out) and multiple-increase (zoom in) approach is used for clients with a high-bandwidth network connection. The closest neighbor set of each file (based on semantic distance and their access history) is used to choose corresponding files for zooming in/out. If all the files belonging to a directory are zoomed out to shadow cache, the whole directory will be zoomed out, too. As such, the dynamic zooming in/out protocol provides the Cegor file system with the ability to adapt to different network connections.

2.3.4. Caching Replacement Algorithm In Cegor, the whole local disk can be used as a cache to hold the temporary files, making the cache size effectively infinite and the cache replacement less important. However, the ever-increasing performance gap between memory and disk drives the adoption of in-memory cache, which is very important to file operations. We propose a semantic-based cache replacement algorithm in the Cegor system [30]. The basic idea is to catch the inter-file relations and intra-file relations.

2.4. Type-Specific Communication Latency Optimization

In addition to reducing the number of messages, we propose an adaptive type-specific differencing scheme to reduce the inherent communication latency associated with each update message in Cegor.

A differencing algorithm computes the difference between two files, ideally producing a small "delta". Several projects have used differencing to build systems that reduce network bandwidth requirements by exploiting commonality between files, such as delta-encoding for Web documents [21] and email files [6], and block-based techniques for file synchronization [23, 35] and distributed shared memory systems [4, 11].

The rationale of our approach is to treat different types of files with different differencing techniques. This idea is based on two observations: (1) previous results show that the communication latency can be optimized by using different differencing techniques, and (2) the files in a file system usually have a variety of characteristics, such as program source codes, email, microsoft documents, images, and latex files etc. We motivate our work by briefly describing three representative algorithms.

Delta-encoding ($Delta$) was first proposed by Mogul *et al.* in the context of HTTP traffic [21]. This approach could dramatically reduce network traffic in cases where a client and a server shared a past version of a Web page. *Fix-sized blocking ($Block_{fix}$)* was used in the Rsync [35] software to synchronize different versions of the same data. In this approach, files are updated by dividing both files into fix-sized chunks. The client sends digests of each chunk to the server, and the server responds only with new data chunks. *Vary-sized blocking ($Block_{varied}$)* was proposed in LBFS [23] for further reducing traffic. The idea behind LBFS is that of content-based chunk identification. Files are divided into chunks, demarcated by points where the Rabin fingerprint [29] of the previous 48 bytes matches a specific value. This tends to identify a portion even after insertions and deletions have changed its position in the file. LBFS then computes a digest of each chunk and sends them to the server, which maintains a database of chunks from all local files. The server responds to the client with bits for new chunks. The client then sends new chunks to the server.

Although both fix-sized and varied-sized blocking allow the system to identify bandwidth saving similarity across (arbitrary) files, its three-way chunk negotiation protocol results in extra overhead, which motivates us to propose a

general client-side approach to eliminate the overhead of the three-way protocol. In our approach, when a modified file wants to be sent back to the server, the client always creates a difference between the last version and this version using a specific differencing algorithm best suited for this kind of files. The difference will then be sent to the server instead of the whole new file. In comparison to the LBFS and Rsync, the client-side approach would dramatically reduce the number of messages between clients and servers.

There are three factors that determine the success of differencing mechanisms: *file similarity identification*, *computation overhead*, and *delta size (communication overhead)*. Several projects have already found that there are great similarity between files in a file system [6, 19], even for files that are not directly related. In our approach, to make the operations of client-side differencing completely local, the last version of this file, which is held at both sides, will always be chosen for difference computation.

We build a simulator to compare the efficiency and overhead of the above three algorithms, with five different type of inputs: source code (`source`), images (`image`), Web document (`web`), Microsoft word documents (`doc`), and latex files (`latex`). Figure 1 shows the comparison of computing overheads and bandwidth savings of four algorithms. For comparison purposes, the corresponding overhead of direct communication (with compression/decompression) is shown in Figure 1 as well, denoted by `direct`.

Regarding the latter two factors, our preliminary results show that different algorithm have different computation overhead and bandwidth saving for different type of files. For example, `delta` is the best optimization algorithm for Web documents. Fingerprint-based vary-sized blocking technique has the potential to reduce the bandwidth requirement for most documents except images, but it always has large computing overhead. Therefore, when the network is very slow this algorithm is useful so that the bandwidth saving can amortize its high computing overhead. In our preliminary work [17], we evaluated the total latency of these algorithms in the context of different network, including dial-up, modem, wireless network, and wired LAN. We found that it is necessary to have an adaptive strategy for communication optimization in a distributed file system.

2.4.1. Similarity Identification The key to the success of client-side differencing algorithms described above is similarity identification, i.e., how to identify files similar to a write back file candidate. Our solution is to classify files into five different categories (as shown in Figure 1) based on their characteristics, and handle them in the specific appropriate differencing algorithm. The semantic view based protocol proposed in Section 2.3.1 guarantees that the last version of each file is always available in both sides in Cegor system. For those files with no previous version and are missing in the first access, a recipe-based protocol will be applied [34].

One of the problems of using versioning is in determining how long to keep individual versions. In our design, although only the last version need be kept, the versioning problem comes from the fact that multiple clients may have different version views of different files. Therefore, multiple versions for the same file maintained by the server should be clean up periodically by a garbage collector based on the reconcile consistency.

2.5. Fault Tolerance and Scalability

Unlike NFS, Cegor introduces states at the server side, such as the last connection view credential and the last semantic view of each client. By doing so, fault tolerance becomes an important concern in our design. If we examine the design of Cegor carefully, we find that most of the states maintained by Cegor are *soft-state*. That is, even those states are lost (e.g., reboot after server crash), the whole system keeps running with a little latency penalty during restarting. For example, if the server loses the last semantic view of one client, the client can generate a complete view (instead of a difference between current view and the last view) and send it to the server. After receiving the complete view, the server checks all the files and directories included in the semantic view against their current states on the server, and sends back a view difference to the client. Furthermore, the reconcile consistency model releases the server from keeping the copy of all callback promises [31] in Cegor, and makes it more fault tolerant than AFS.

The objective of Cegor is to provide consistent adaptive file access to either a single person or a small collaborative group. The mechanisms that we have discussed should help the server to scale to medium numbers of users as well. For example, the soft state of server design, semantic-view based caching, and type-specific differencing algorithm all help reduce the load of the server. Cegor can also automatically benefit from the server replication techniques to address availability and scalability, such as xFS [2].

3. Related Work

There is a great deal of prior work in distributed and wide-area file systems, and is too much to be surveyed here [3, 10, 14, 23] and connection mobility [18, 33]. Instead of discuss all of them, we concentrate on those projects that share one or more of the following goals with Cegor: *authentication*, *wide-area file systems*, *disconnected operation*, *caching and prefetching*, and *communication optimization*.

Authentication Authentication between (remote) clients and file servers has been addressed in several systems [3,

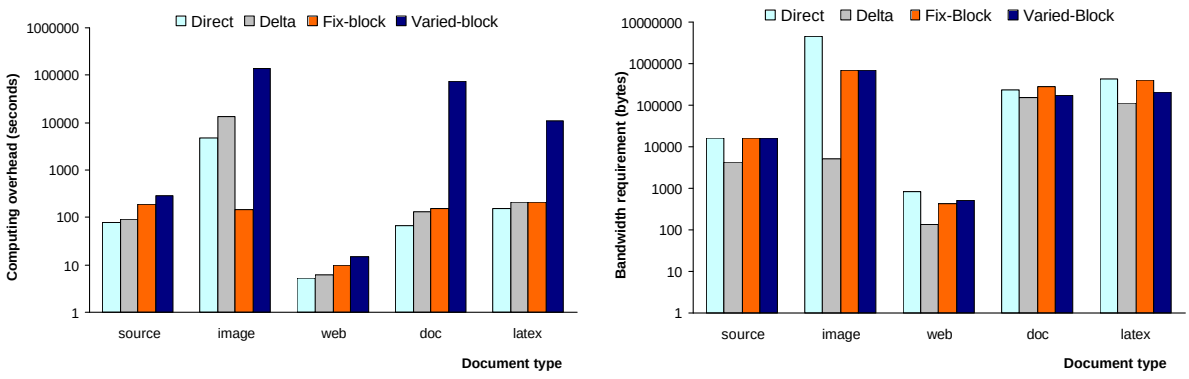

Figure 1. A comparison of different differencing algorithms over five difference types of documents: (a) computing overhead, (b) bandwidth requirements.

10]. NFS provides security by relying mainly on authenticating RPCs. Security in AFS is based on Kerberos [24], which requires that an AFS user must be part of the same Kerberos realm as the remote file server, thus limiting its deployment on wide-area settings. Cegor proposes to use view credentials to authentication users, and use file access keys to control the file access permissions.

Wide-area File Systems AFS [10], LBFS [23], Fluid replication [13], CASPER [34] and SFSRO [7] are the closest systems to Cegor in terms of the intended environment. Most of the previous systems focus on scalability, security and/or availability issues by self-managed replication and caching. However, Cegor takes the adaptability as the first-class design option. Cegor's type-specific communication optimization is motivated by LBFS's idea of sharing blocking among multiple files to reduce bandwidth requirements, but Cegor extends this idea to handle files in a type-specific way.

Disconnected Operation Disconnection and weakly connection have been addressed in several file systems, such as Coda [15], Bayou [28], and Ficus [27]. Although different in the design details, these systems share the idea of supporting localized operations during disconnection by accurately hoarding files to local machine as much as possible. Unlike these previous efforts, the goal of Cegor is to support clients who have continuous "always on" but heterogeneous connections. Furthermore, system support for transparent reconnection and conflict resolution is an important feature proposed in Cegor.

Communication Optimization Rsync [35] synchronizes different versions of the same data by using fix-sized chunks for communications. LBFS [23] takes this a step further by reusing the data chunks across multiple similar files (including multiple versions of the same file). As described in Section 2.4, Cegor classifies the data into several categories and adopts different optimization techniques for storage and communication optimizations. Also, Cegor's adaptive approach considers the tradeoff between communication and computation.

4. Current Status and Future Work

In this paper, we proposed to build an adaptive distributed file system which provides the "*close and go, open and resume*" semantics across heterogeneous network connections, ranging from high-bandwidth local area network to low-bandwidth dial-up connection. Our approach relies on a set of new techniques for managing adaptive access to remote files, including three components: *system support for secure, transparent reconnection* at different places, *semantic-view based caching* to reduce communication frequencies in the system, and *type-specific communication optimization* to minimize the bandwidth requirement of synchronizations between clients and servers.

Currently, we have implemented several key parts of the whole file system, such as home-based authentication protocol [32], semantic-distance based cache replacement [30], and adaptive communication optimization [17]. Our next step is integrating these modules together into a Cegor prototype, and evaluated in a controlled emulator-based environment using Andrew Benchmark [10] and distributed file traces [22, 30] at the first stage. We also plan to conduct a comprehensive comparison between Cegor and other start-of-the-art distributed file systems, such as NFS [3], AFS [10], CODA [15], and LBFS [23].

References

[1] F. 180-1. *Secure Hash Standard*. U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, VA, Apr. 1995.

[2] T. Anderson, M. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Wang. Serverless network file systems.

ACM Transactions on Computer Systems 14(1):41–79, Feb. 1996.

[3] B. Callaghan and P. Staubach. NFS Version 3 Protocol Specification, rfc 1813, June 2000.

[4] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Techniques for reducing consistency-related communication in distributed shared memory systems. *ACM Trans. on Computer Systems* 13(3):205–243, Aug. 1995, `http://www.cs.utah.edu/~retrac/papers/tcos95.ps.Z`.

[5] Citrix Inc., `http://www.citrix.com`.

[6] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. *Proc. of the USENIX 2003 Annual Technical Conf.*, June 2003.

[7] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems* 20(1), Feb. 1992.

[8] P. S. Henry and H. Luo. Wifi: What's next? *IEEE Communications Magazine* 40(12):66–72, Dec. 2002.

[9] A. Hisgen, A. Birrel, T. Mann, M. Schroeder, and G. Swart. Availability and conistency tradeoffs in the echo distributed file system. *Proceedings of the 2nd Workshop of Workstation Operating Systems*, pp. 49-54, Sept. 1989.

[10] J. H. Howard, M. Kazar, S. Menees, d. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems* 6(1):51–81, Feb. 1988.

[11] W. Hu, W. Shi, and Z. Tang. Optimizing home-based software dsm protocols. *Cluster Computing: The Journal of Networks, Software and Applications* 4(3):235–241, 2001, `http://www.cs.wayne.edu/~weisong/papers/cc.pdf`.

[12] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable seure file sharing on untrusted storage. *Proc. of the 2nd USENIX Conf. On File and Storage Technologies*, pp. 29-41, Apr. 2003.

[13] M. Kim, L. Cox, and B. D. Noble. Safety, visibility, and performnace in a wide-area file systems. *Proc. of the 1st USENIX Conf. On File and Storage Technologies*, Jan. 2002.

[14] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *Proc. of the 13nd ACM Symp. on Operating Systems Principles*, Oct. 1991.

[15] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Code File System. *ACM Transactions on Computer Systems* 10(1), Feb. 1992.

[16] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)*, Oct. 1997.

[17] H. Lufei and W. Shi. Performance evaluation of communication optimization techniques for distributed file systems. Tech. Rep. MIST-TR-2003-006, Department of Computer Science, Wayne State University, July 2003.

[18] D. Maltz and P. Bhagwat. MSOCKS:an architecture for transport layer mobility. *Proc. of the IEEE INFOCOM'98*, 1998.

[19] U. Manber. Finding similar files in a large file system. *Proceedings of the USENIX Winter 1994 Technical Conference*, pp. 1-10, Jan. 1994.

[20] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. *Proc. of the 17th ACM Symp. on Operating Systems Principles (SOSP-17)*, pp. 124-139, Dec. 1999.

[21] J. C. Mogul, F. Douglis, a. Feldmann, and B. Krishnamurthy. Potential Benefits of Delta-Encoding and Data Compression for HTTP. *Proc. of the 13th ACM SIGCOMM'97*, pp. 181-194, Sept. 1997, `http://www.douglis.org/fred/work/papers/sigcomm97.pdf`.

[22] L. B. Mummert and M. Satyanarayanan. Long term distributed file reference tracing: Implementation and experience. *Software-Practice and Experience* 26(6):705–736, 1996.

[23] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, Oct. 2001.

[24] B. C. Neumann and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine* 32(9):33–38, Sept. 1994.

[25] The OpenAFS Project, `http://www.openafs.org`.

[26] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. nelson, and B. B. Welch. The sprite network operating system. *IEEE Computer* 21(2):23–36, Feb. 1988.

[27] T. W. Page, R. G. Guy, J. s. Heidemann, D. Ratner, P. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software-Practice and Experience* 28(2):123–133, Feb. 1998.

[28] K. Peterson, M. J. Spreitzer, D. Terry, and M. Theimer. Bayou: Replicated database services for wide-wide applications. *Proceedings of the 7th ACM SIGOPS European Workshop*, Sept. 1996.

[29] M. O. Rabin. Fingerprinting by random polynomials. Tech. Rep. TR-15-81, Harvard Aiken Computation laboratory, 1981.

[30] S. Santhosh and W. Shi. Semantic-distance based cache replacement for wide-area file systems. Tech. Rep. MIST-TR-2004-002, Department of Computer Science, Wayne State University, Jan. 2004.

[31] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer* 23(5), May 1990.

[32] W. Shi and S. Santhosh. Home-based authentication protocol for nomadic users. *Proceedings of the 2003 International Conference on Internet Computing*, June 2003.

[33] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. *Proc. of the 6th ACM SIGMOBILE International Conference on Mobile Computing and Networking (MobiCom'00)*, pp. 155-164, Aug. 2000.

[34] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, and A. Perrig. Opportunistic use of content addressable storage for distributed file systems. *Proc. of the USENIX 2003 Annual Technical Conf.*, June 2003.

[35] P. Tridgell and P. Mackerras. The rsync algorithm. Tech. Rep. TR-CS-96-05, Department of Computer Science, Australian National University, 1996.