SEARS: Space Efficient And Reliable Storage System in the Cloud

†Ying Li, ‡Katherine Guo, †Xin Wang, ‡Emina Soljanin, ‡Thomas Woo
†Dept of Electrical and Computer Engineering, Stony Brook University ‡Bell Labs, Alcatel-Lucent

***Abstract*—Today's cloud storage services must offer storage reliability and fast data retrieval for large amount of data without sacrificing storage cost. We present SEARS, a cloud-based storage system which integrates erasure coding and data deduplication to support efficient and reliable data storage with fast user response time. With proper association of data to storage server clusters, SEARS provides flexible mixing of different configurations, suitable for real-time and archival applications.**

Our prototype implementation of SEARS over Amazon EC2 shows that it outperforms existing storage systems in storage efficiency and file retrieval time. For 3 MB files, SEARS delivers retrieval time of 2.5 s compared to 7 s with existing systems.

***Keywords*—cloud, storage, deduplication, erasure coding**

I. Introduction

Data from connected devices today are flowing into data centers with an unprecedented rate. More than half of the companies in the survey of global enterprise market currently store at least 100 TB of data and one-third expect their data to double in the next two to three years [3].

The cloud infrastructure enables low-cost and scalable file storage that provides global file access. Any file system must offer reliable storage whether through file duplication that requires more space but less computation complexity such as GFS [8] or through erasure coding that requires less space but more computation complexity such as RAID systems [5]. At the same time, raw data exhibit redundancy across files. This redundancy can be explored to reduce storage cost mainly for backup systems [11], [14], [17]. Using these techniques, data are divided into chunks and unique data chunks are stored once and referenced multiple times. Different from archival systems, cloud-based storage systems are required to support interactive user access with reasonable response time.

We propose a cloud-based file system named **SEARS**-Space Efficient And Reliable Storage system that exploits the deduplication technique to reduce storage and traffic cost as well as the erasure coding technique to increase both the data reliability and the file retrieval speed. Given a file, there are different ways to associate data chunks with available storage servers and retrieve data. Archive-based backup systems mainly care about storage efficiency and reliability. However, interactive cloud storage systems also care about file retrieval speed. To meet different application needs, we propose two data-server binding schemes with different performance goals: (1) faster file access speed or (2) higher storage efficiency.

We aim for SEARS to serve as a reference design for a flexible cloud storage framework that can support customized level of deduplication, modes of coding and server binding, and the mix of different modes. Its flexibility handles different application scenarios, from batch-centric archival to real-time.

Related Work: Recent studies have reported that erasure coding can guarantee the same level of content accessibility with lower storage than replication [9], [10], [16].

File deduplication relies dividing files into chunks and eliminating the need to store or transfer identical chunks multiple times. LBFS [14] introduced content based chunking with Rabin fingerprints [15]. Various work improves on the idea by compressing data chunks [17], comparing chunks belonging to highly related files [11], switching between large and small chunk sizes to discover more overlapping regions [13].

The tradeoff between increasing data reliability and deduplicating data poses great challenge to the design of a reliable and space efficient storage system. The archival storage system R-ADMAD [12] combines the content-based chunk deduplication with erasure coding. However, it uses large and fixed-size data chunk containers, which reduces deduplication efficiency and makes it ill-suited for delay-sensitive applications.

II. SEARS Architecture

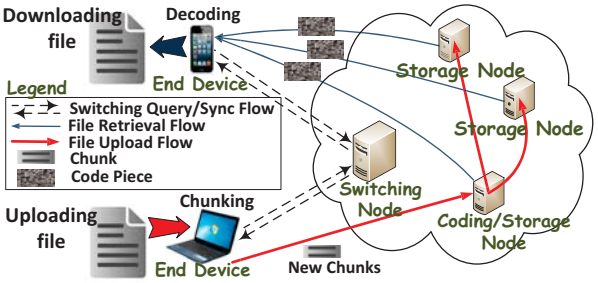


Fig. 1. SEARS system overview. One end device (laptop) uploads a file where the file is chunked at end device and the meta-data for the file is uploaded to the **switching node** for the user. Unique chunks for the file missing from SEARS are sent to and coded at the **coding node** which is one of the server nodes in the cluster storing code pieces of the chunks for the file. Another end device (smart phone) downloads a file where code pieces of each unique chunk are retrieved from multiple storage nodes in SEARS concurrently.

Figure 1 shows the SEARS system architecture consisting of storage server nodes operating in a data center. Users use SEARS as any file system by storing (or uploading) files to server nodes; and retrieving (or downloading) files from server nodes. Each user accesses SEARS through a designated storage server node we call **switching node** for the user and all the user's files. Each user end device is configured with host name or the IP address of the the user's switching node since it

978-1-4673-6770-7/15/$31.00 ©2015 IEEE

is the first node to reach SEARS. We consider a total of N nodes in SEARS divided into non-overlapping **clusters** of size n. The reason of forming cluster of nodes is due to the need of storing coded chunks at multiple nodes for reliability. We assign each cluster with a unique **cluster id**. We focus on the single data center configuration in this work. However, the concept of SEARS can be naturally extended to multiple data centers.

Content-based Chunking Operation: Before storing data, SEARS first removes redundant content. Files are divided into chunks and unique chunks are stored only once. We use content-based chunking to better capture redundancy [7]. Using smaller chunk size can result in more duplicate chunks thus achieving higher levels of deduplication. However, it also results in larger number of chunks and therefore larger overhead in meta-data management and reduced system performance. Furthermore, disk operations benefit from continuous data access, while smaller chunks lead to less efficient random access pattern. To balance the tradeoff, we choose average chunk size of 4 KB [6] [14] and enforce the minimum and maximum chunk sizes to be 1 KB and 8 KB respectively. For each chunk, we apply the 160-bit SHA-1 hash function [6] to generate a fixed-size hash value to serve as the **chunk id**.

File Storage Operation: Ahead of data storage, SEARS explores both intra-file and inter-file content redundancy and eliminates all redundant content. In the first step, SEARS eliminates intra-file redundancy as follows. Before a user file is uploaded into SEARS, the end device applies content-based chunking to the file, and generates chunk id for each chunk, and produces **file chunk-meta-data** for the file, which is composed of a sequence of entries for all chunks in the file and each entry consists of a chunk id and a cluster id specifying the cluster that stores the chunk. The file chunk-meta-data is stored at (1) the user's end device and uploaded to (2) the SEARS switching node serving the user. After this process, only non-repeating chunks will be kept so that intra-file redundancy can be eliminated.

A file in SEARS is represented by its file chunk-meta-data. Each unique chunk is stored as n code pieces in an n-node cluster. The user's switching node keeps a **chunk-meta-data-table** that stores one file chunk-meta-data for each file belonging to the user. As a chunk can appear in multiple files, we define the **reference count** for a chunk as the number of files in SEARS that the chunk appears in. The chunk reference count is updated as SEARS evolves with file addition, removal and update.

In the second step, SEARS eliminates inter-file redundancy across the set of nodes responsible for storing the file as follows. The user's switching node in SEARS removes chunk ids already in the set of nodes and forms a list of ids of missing chunks for the end device to upload directly to the set of storage nodes. This means only unique chunks that are not present in the set of SEARS nodes are uploaded from the user's end device. As a result, bandwidth between the user's end device and SEARS is only required to transfer non-redundant data.

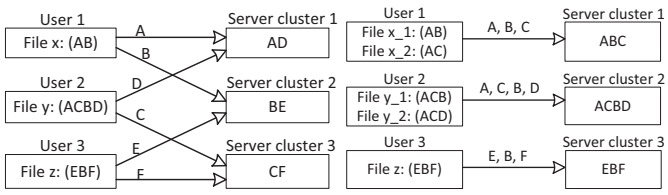
File Retrieval Operation: Whenever an end device retrieves a file from SEARS for the first time, the requesting end device does not have the file chunk-meta-data and the retrieval request is sent to the user's switching node. The switching node first sends back the file chunk-meta-data. The end device then checks the list of chunk ids in the file chunk-meta-data against the list of chunk ids already in its local storage, and determines the missing chunks needed to construct the file. The end device then only requests the missing chunks from SEARS.

File Chunk-Meta-Data Synchronization Operation: In the case when the end device and its responsible switching node in SEARScloud each has a version of the file chunk-meta-data, synchronization is required to resolve any conflicts. We follow the policy for the copy with the latest time-stamp to overwrite the one with an earlier time-stamp. We assume clock synchronization between the user's end device and SEARS is provided with mechanisms such as NTP [4].

Erasure Coding and Decoding Process: In SEARS, each unique chunk first reaches a node in the cluster that stores the code pieces of the chunk, we call **coding node**. The coding node then divides the chunk into k equal-sized pieces and codes it into n code pieces through (n, k) erasure coding with $n \geq k$. These n code pieces are associated with a cluster of n storage nodes and exactly one piece is stored in one node in the cluster. Note that any node in the cluster can serve as the coding node for a chunk to be stored at the cluster.

Whenever the user's end device requests a missing chunk in a file based on the file chunk-meta-data, it issues n concurrent requests to the n nodes in the cluster identified by the cluster id and as soon as k code pieces are received, it reconstructs the chunk and terminates any ongoing connection to the remaining $n - k$ nodes. This design benefits from parallel download of data to reduce SEARS response time as we show in Section IV.

III. Server Binding Schemes



(a) Chunk Level Binding (CLB) (b) User Level Binding (ULB)

Fig. 2. Illustration of the binding schemes.

Consider SEARS nodes grouped into M clusters of size n. A file to be stored in SEARS is divided into chunks and each chunk is coded into n code pieces to be stored in a cluster. A key design question for SEARS is to determine how to associate data to clusters. We call this the **binding** process. Different applications have different requirements for cloud-based storage services, including fast file retrieval, small space usage in order to reduce storage cost. We design binding schemes across the spectrum of application requirements namely Chunk Level Binding and User Level Binding with examples in Figure 2(a) and 2(b) respectively.

Chunk Level Binding (CLB): For archival applications that runs in the background and demands storage efficiency, the binding process must offer system wide data deduplication. The **Chunk Level Binding (CLB)** scheme selects the best cluster to store each chunk. CLB is ideal for large media content repository like YouTube and NetFlix where users share the same or similar content. Each unique chunk entering SEARS is assigned to a cluster such that storage space of all clusters are evenly consumed as time passes. Note that all storage and retrieval requests must pass through the user's switching node. To distribute load evenly to clusters, we use a greedy algorithm to assign a chunk to the cluster with the largest amount of free storage space.

User Level Binding (ULB): For interactive applications with emphasis on promptness of file retrieval, the binding scheme must offer simplicity in chunk retrieval. The **User Level Binding (ULB)** scheme binds each user with a fixed cluster and simplifies file retrieval process as all chunks of this user are stored in the same cluster. Initially each user is assigned a fixed cluster. When storage capacity is exhausted at the cluster assigned for the user, a new cluster is assigned to future files from the user. This is equivalent to assigning a subset of user files to a separate user and only intra-set redundancy within the subset of files can be captured. ULB incurs at most one extra cluster id for a subset of user files, offers simple retrieval process but sacrifices space efficiency, as the chunks stored in different clusters belonging to different users (or even the same user) can not be exploited globally during the deduplication process.

The two binding schemes described so far offer different tradeoffs in space saving and file retrieval response time. However, they are just examples to showcase the flexibility in the design of SEARS . We design SEARS to be a powerful platform that use both deduplication and erasure coding in the best combination to fit various application needs.

IV. Performance Evaluation

We evaluate the performance of our prototype implementation of SEARS over Amazon EC2 [1]. We generate a data set reflecting real-time data access of 10 users during a span of 3 weeks in 2014 containing three parts. (1) User Personal Data of 1.6 TB consisting of various common types of files from 10 users; (2) System Log of 132 GB consisting of major system log files (e.g. files under $/var/log$ directory) of Amazon EC2 Ubuntu server machines recorded every hour; and (3) System Backup Image of 3.5 TB consisting of the complete backup image files for Linux systems created once a day.

We evaluate SEARS in terms of storage usage with deduplication ratio and time performance with the average file retrieval time. **Deduplication Ratio** is defined as the ratio of the total size of original files over the total space consumption for SEARS including the indexing overhead for storing them. This metric captures the combined effect of deduplication (reduce space usage) and erasure coding (increase space usage). **Average File Retrieval Time** is defined as the average time duration from the moment the user issues a request for a file to the moment the file is ready at end device. This involves downloading and decoding of all necessary chunks and reconstruction of the file from all chunks.

We employ 10 Amazon EC2 instances as driver machines to generate the log files, system backup images in addition to making users upload their own personal data. We fix cluster size at $n = 10$ thus use 10 EC2 instances for each cluster. We use $E = 20$ clusters.

We compare SEARS with the existing storage system R-ADMAD [12] which packs variable-length data chunks into fixed size objects of 8 MB which are encoded with erasure code and distributed among storage nodes called redundancy groups. To fairly evaluate R-ADMAD with SEARS, we implement it on EC2 cloud, and follow the same chunking process as SEARSas specified in Section II for all files in our data set to generate chunks of 4 KB average size. Furthermore, the same set of nodes are used for the SEARS cluster and the R-ADMAD redundancy group.

Effect of k/n Ratio: The ratio k/n has profound performance impact on any scheme using erasure coding. To illustrate this, we fix n at 10 and vary k for the data set. As each chunk requires n/k times as much space as before the coding process, deduplication ratio increases with k as shown in Figure 3(a). Increases of k also lead to larger numbers of code pieces with smaller sizes for each chunk. This implies more parallel retrieval processes, each with smaller bandwidth requirement. With smaller k ($k < 5$), both factors contribute to reduced chunk and file retrieval time. However, after k increase beyond a threshold, $k = 5$ for the data sets, the larger number of concurrent retrieval processes and the decoding process with more code pieces become the bottleneck and increase retrieval time as shown in Figure 3(b). CLB exploits redundancy across all chunks in all files and achieves a higher deduplication ratio. However, the process of searching for chunks across all clusters leads to the higher file retrieval time. On the other hand, ULB can only exploit intra-user redundancy which leads to a lower redundancy ratio. However all chunks in a file are easily retrieved from one cluster, which leads to the faster file retrieval time. We use $k = 5$ and $n = 10$ from now on.

Deduplication Ratio: To see how the ratio changes as data volume evolves over time, we plot the cumulative deduplication ratio on the 5^{th}, 10^{th}, 15^{th}, and 21^{st} day in Figure 3(c). The ratio improves for all schemes over time as data volume increases, for more redundancy can be exploited. It also shows deduplication ratio decreases in the order of CLB, R-ADMAD, and ULB. R-ADMAD is essentially same as CLB in data deduplication as it can exploit system wide redundancy just as CLB. But R-ADMAD uses slightly more space than CLB because of its indexing structure is more complex than CLB.

Time Performance: To examine interactive user experience, we replay the request pattern captured in the user personal data trace of our data set. We use 10 desktop machines residing in the eastern region of the US. Each desktop replays the file access trace for each of the 10 users. We report the file retrieval time for files accessed during each hour of the day averaged over 21 days over 10 users. To retrieve a file, the user's end

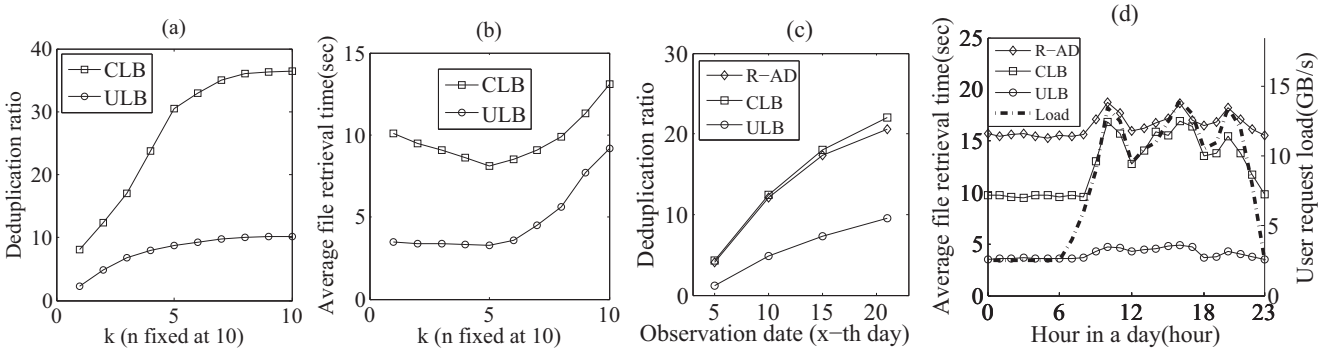


Fig. 3. (a) k/n effect on Dedup ratio; (b) k/n effect on retrieval time; (c) Dedup ratio; (d) file retrieval time

device directly requests data chunks from 10 nodes storing the code pieces of each chunk in the three schemes. Figure 3(d) presents file retrieval time in relation to user request load averaged over each hour of the day over 21 days. Users' data request volume per hour in these figures reflect work activity during a day, that is, light activity at night (0:00 midnight to 8:00 am) and heavy and fluctuating activity for the rest of the day. ULB offers the fastest and relatively flat retrieval time because requests from the same user are handled by one cluster and there are no multiple requests for the same data chunk at the same time. CLB offers slower file retrieval than ULB, and large fluctuation during the working hours closely matching data request volume. This is because a unique chunk is stored only once in the entire system, and multiple users can request the same unique chunk at the same time, which leads to congestion at the cluster hosting the chunk in demand. R-ADMAD follows the data volume fluctuation during the day but with larger retrieval time than SEARS.

To compare with a commercial system, we note that downloading 3 MB files from the same set of 10 desktops residing in the eastern part of the US takes an average of 7 s from Amazon EC2 service in us-east-1 region [2]. With ULB in SEARS, the download time is 2.5 s throughout the day.

V. Conclusion and Future Work

We describe the design and implementation of a space efficient, data reliable and fast retrieving cloud-based storage system SEARS which integrates data deduplication and erasure coding. SEARS provides a flexible combination of various binding schemes to associate server nodes with data to be stored at different level based on application needs. Evaluation over Amazon EC2 shows that SEARS outperforms related systems with lower storage usage while ensuring fast and reliable data access.

As future work, we plan on examining the location of cluster nodes inside data centers to future improve data reliability and reduce retrieval time. We are evaluating the system with more data sets with additional metrics such as storage balance, file upload time and file retrieval success rate. Various system design parameters in SEARS and performance under flexible configuration of SEARS with multiple binding schemes, chunk size and erasure codes also need further investigation.

References

[1] Aws ec2. In *http://aws.amazon.com/ec2*.

[2] Cloud match. In *https://cloudharmony.com/speedtest*.

[3] Global enterprise big data trends:2013. In *http://www.microsoft.com/en-us/news/download/presskits/bigdata/docs/bigdata_021113.pdf*.

[4] Network time protocol. In *RFC 1305*.

[5] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. Raid: High-performance, reliable secondary storage. *ACM Comput. Surv.*, 26(2), 1994.

[6] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 2–2, 2011.

[7] D. Frey, A.-M. Kermarrec, and K. Kloudas. Probabilistic deduplication for cluster-based storage systems. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 17:1–17:14, 2012.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, 2003.

[9] G. Joshi, Y. Liu, and E. Soljanin. On the delay-storage trade-off in content 1 download from coded distributed storage systems. *IEEE J. Select. Areas Commun.*, 2014.

[10] L. Huang, S. Pawar, H. Zhang and Kannan Ramchandran. Codes can reduce queuing delay in data centers. *Proc. Int. Symp. Inform. Theory*, pages 2766–2770, Jul. 2012.

[11] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 111–123, Berkeley, CA, USA, 2009. USENIX Association.

[12] C. Liu, Y. Gu, L. Sun, B. Yan, and D. Wang. R-admad: high reliability provision for large-scale de-duplication archival storage systems. In *ICS*, 2009.

[13] G. Lu, Y. Jin, and D. H. C. Du. Frequency based chunking for data de-duplication. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 287–296, Washington, DC, USA, 2010. IEEE Computer Society.

[14] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 174–187, 2001.

[15] M. Rabin. Fingerprinting by random polynomials. In *Harvard University, Technical Report*, pages TR–15–81, 1981.

[16] U. Ferner, M. Médard, and E. Soljanin. Toward sustainable networking: Storage area networks with network coding. *Allerton Conf. on Commun. Control and Comput.*, Oct. 2012.

[17] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.