Cost-Efficient Remote Backup Services for Enterprise Clouds

Yu Hua, *Senior Member, IEEE*, Xue Liu, *Member, IEEE*, and Dan Feng, *Member, IEEE*

***Abstract*—In order to efficiently achieve fault tolerance in cloud computing, large-scale data centers generally leverage remote backups to improve system reliability. Due to long-distance and expensive network transmission, the backups incur heavy communication overheads and potential errors. To address this important problem, we propose an efficient remote communication service, called Neptune. Neptune efficiently transmits massive data between long-distance data centers via a cost-effective filtration scheme. The filtration in Neptune is interpreted as eliminating redundancy and compressing similarity of files, which are generally studied independently in the existing work. In order to bridge their gap, Neptune leverages chunk-level deduplication to eliminate duplicate files and uses approximate delta compression to compress similar files. Moreover, in order to reduce the complexity and overheads, Neptune uses a locality-aware hashing to group similar files and proposes shortcut delta chains for fast remote recovery. We have implemented Neptune between two data centers and their distance is more than 1200 km via a 2 Mb/s network link. We examine the Neptune performance using real-world traces of Los Alamos National Laboratory (LANL), EMC, and Campus collection. Compared with state-of-the-art work, experimental results demonstrate the efficiency and efficacy of Neptune.**

***Index Terms*—Backup systems, cloud storage, compression, reliability.**

I. Introduction

MANY INDUSTRIAL cloud computing applications require high degree of reliability and availability; hence, data centers and their backups are usually built in a geographically dispersed manner, to prevent failures from disasters, such as earthquakes, tsunami, and hurricanes. The unpredictable occurrence of disasters may destroy the entire datasets stored in a data center, e.g., as a result of severe network outages during super storm Sandy. Therefore, large-scale cloud networks generally rely on regular remote backups to protect against the disasters. In general, long-distance network connectivity is expensive and/or bandwidth constrained, making remote backups for massive data very costly in terms of both network bandwidth and backup time.

Manuscript received September 30, 2014; revised September 10, 2015; accepted February 19, 2016. Date of publication March 17, 2016; date of current version October 31, 2016. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61173043 and in part by the State Key Laboratory of Computer Architecture under Grant CARCH201505. Paper no. TII-15-1832.

Y. Hua and D. Feng are with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: csyhua@hust.edu.cn; dfeng@hust.edu.cn).

X. Liu is with the School of Computer Science, McGill University, Montreal, QC H3A 0E9, Canada (e-mail: xueliu@cs.mcgill.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2016.2543258

An intuitive and direct solution is to detect data redundancy in the backup data stream to reduce the amount of data actually transmitted. It is worth noting that, from a systems-implementation perspective, it is important to distinguish between the managed and the unmanaged redundancy. The former is purposely introduced by the system to support and improve availability, reliability, and load balance. The latter, however, is a property of the data itself and thus invisible to the system. Due to its consumption of substantial system resources, the unmanaged redundancy can become a potential performance bottleneck in cloud systems. The cost effectiveness and efficiency of remote backup lie in the significant improvements on the effective backup throughput, which can be achieved by transmitting data difference in a compressed manner.

Compression techniques are widely used in computer networks and enterprise clouds to increase the efficiency of data transfers and reduce bandwidth requirements. Most techniques focus on the problem of compressing individual files or data streams. However, files in the enterprise clouds are often replicated and frequently modified. Hence, the enterprise clouds are full of many redundant data. For example, the receiver in a data transfer already contains an earlier version of the transmitted file or similar files, or multiple similar files are transmitted together. These redundant data can be deduplicated or delta compressed. Delta compression techniques are concerned with efficient file transfer over a slow communication link in the case where the receiving party already has a similar file.

Since the network bandwidth across cloud systems is often a performance-limiting factor, existing systems leverage data reduction techniques to reduce the unmanaged redundancy and improve the effective throughput. The most commonly used techniques include chunk-level deduplication and delta compression, whose goal is to prevent redundant data from being transferred. Deduplication schemes split files into multiple chunks (say, generally 8 kB size), where a hash signature, called a fingerprint, uniquely identifies each chunk. By checking their fingerprints, duplicate chunks can be removed, while avoiding a byte-by-byte comparison and replacing identical data regions with references. Moreover, the delta compression compresses similar regions by calculating their differences. The efficiency of remote backups depends on exploring and exploiting the property of redundancy.

The data filtration in Neptune needs to compute a sketch of each nonduplicate chunk as a similarity measure. Sketches have the property that if two chunks have the same sketch, they are likely near duplicates. These can be used during backups to identify similar chunks. Moreover, instead of using a full index mapping sketches to chunks, Neptune loads a cache with sketches from a previous stream to obtain close compression performance to a full sketch index. For a backup, identical chunks are deduplicated, and nonduplicate chunks are delta compressed relative to similar chunks that already reside at the remote repository. We then compress the remaining bytes and transfer across the WAN to the repository. Specifically, this paper has the following contributions.

1) Comprehensive Filtration: Neptune offers comprehensive filtration between the source and the destination of a remote backup. In the source, Neptune eliminates duplicate files and compresses similar files. The remote transmission leverages a similarity detection technique to obtain significant bandwidth savings. Neptune goes far beyond the simple combination of system-level deduplication and application-level similarity detection. While the former can deduplicate exact-matching chunk-level data well, it fails to deal with files from the application's viewpoint, since the low-level chunks cannot explicitly express the properties of application-level data. The latter only concentrates on the files themselves from the application's viewpoint, failing to capture and leverage the system-level characteristics, such as metadata and access patterns. Neptune, in fact, bridges the gap between them and delivers high performance.

2) Cost-Effective Remote Backups: During remote backups, Neptune alleviates computation and space overheads. First, to reduce the scope of processing data, Neptune leverages semantic-aware groups using locality-sensitive hashing (LSH) [1] that has a complexity of $O(1)$ and light space overhead. In order to improve the efficiency of delta compression, Neptune slightly looses the selection of base fingerprint by using top-k, rather than only one similar fingerprint in conventional approaches. The top-k approximate delta compression can identify more chunks to be delta compressed, thus, significantly reducing the entire network overheads. Moreover, in order to support efficient remote recovery, we propose a shortcut scheme for delta chains. The shortcut scheme allows any given version to be restored by accessing at most two files from the version chain. Neptune hence avoids extra computation latency on the intermediate deltas and supports fast recovery.

3) Prototype Implementation and Real-Life Evaluation: We have implemented all components of the Neptune architecture. We built a prototype to compute fingerprints and features, which are stored together in caching units, called storage containers. We used 8-kB chunk size and 4.5-MB containers holding chunks, fingerprints, and features. We examine the performance of Neptune using multiple real-world datasets, including Los Alamos National Laboratory (LANL), EMC, and Campus collection. We also compare Neptune with state-of-the-art work, including EndRE [2], cluster-based deduplication (CBD) [3], and EMC stream-informed delta compression (SIDC) [4].

This paper is organized as follows. Section II presents the related work. Section III presents the Neptune design. We present the experiment setup and evaluation results in Section IV. We conclude our paper in Section V.

II. Related Work

Deduplication schemes split files into multiple chunks. Each chunk is uniquely identified by a hash signature, called a fingerprint. By checking their fingerprints, duplicate chunks can be removed while avoiding a byte-by-byte comparison and replacing identical data regions with references. Recent efforts [5], [6] leverage locality-aware grouping strategy to chunk-level deduplication, which aggregate similar and correlated files into the same or adjacent groups. This strategy can narrow the scope of the files to be probed, but they are costly in capturing the semantics from contents.

Network-wide redundancy elimination is an important research topic and has received many attentions from both academia and industry. EndRE [2] uses an adaptive SampleByte algorithm for fingerprinting and an optimized data structure for reducing cache memory overhead. SIDC [4] proposes an architecture that uses SIDC to already existing deduplication systems without the need of persistent indexes. CBD [3] examines the tradeoffs between stateless data routing approaches with low overhead and stateful approaches with high overhead but being able to avoid imbalances.

Early efforts to reduce the amount of data to be transmitted mainly leverage incremental backup which only detects changes at the granularity of a file. Delta compression transmits the data in the form of differences between current version and the replicated version [7], [8]. The general-purpose delta compression tools are based on the Lempel–Ziv approach [9]. Examples of such tools are vdelta and its newer variant vcdiff [10], the Xdelta compressor [8], and the zdelta tool [7]. In practice, delta compression suffers from the high overheads in terms of computation and network bandwidth. The reason is that delta compression needs to compute the difference between original and new versions, which generally exists in different locations. Hence, the deltas from the chunk-level compression fail to identify similar files in an efficient manner [8].

In order to improve the performance of data synchronization (sync) operation in the cloud, a novel metric, called traffic usage efficiency (TUE) [11], is proposed to quantify the traffic usage efficiency and offer cost-efficient cloud storage services. In order to reduce data sizes in the storage systems, redundancy elimination at the block level (REBL) [12] makes use of compression, duplicate block suppression, and delta encoding, as a hybrid scheme, to delete redundant data in a cost-efficient manner. In order to obtain bandwidth efficiency, tiered approach for eliminating redundancy (TAPER) [13] can synchronize many data across geographically distributed replica locations without the need of prior knowledge of the replica state. Volley [14] performs automatic data placement across geographically distributed data centers while reducing WAN bandwidth costs. Cimbiosys [15] offers a replication platform to allow each device to define its own content-based filtering criteria. By exploiting the skewness in the communication patterns, a tradeoff between improving fault tolerance and reducing bandwidth usage is obtained in [16]. Moreover, update-batched delayed synchronization (UDS) [17] is proposed to batch updates from clients and preserve rapid file synchronization, thus efficiently reducing the maintenance traffic overhead.

Unlike existing work, Neptune judiciously implements the deduplication in local servers and proposes a novel approximate delta compression to obtain significant bandwidth savings. Neptune also leverages shortcut delta chains to support fast remote recovery. Moreover, compared with our previous work [18], the main contributions of this paper include the resemblance detection, delta chains, delta compression, and recovery, as well as the performance evaluation using real-world industrial datasets.

III. Neptune Design

In a communication scenario, both sender and receiver contain a reference file that is similar to the transmitted file. We only need to transmit the difference (or delta) between two files to reduce communication overheads. There are growing interests for supporting redundancy elimination as a network-wide service [19]. The network service can reduce link loads and increase effective network capacity, thus, efficiently meeting the needs of handling the increasing number of bandwidth-intensive applications. In practice, it is difficult and inefficient to leverage existing single-point (one server) redundancy elimination solutions for enterprise clouds due to the lack of the network-aware design. The proposed Neptune is a practical and efficient scheme for network-wide redundancy elimination, which meanwhile improves the utilization of available network resources.

A. Resemblance Detection

In order to delta compress chunks, we need to identify the identical and similar chunks that have been already transmitted. A resemblance sketch can identify the features of a chunk. The sketch-based schemes have been widely used in real-world applications. The sketch will not change even if small variations exist in the data. In order to compute the features in an efficient manner, we use a rolling hash function over all overlapping small regions of data (e.g., 32-byte windows). We choose the maximal hash value as the feature. By using multiple different hash functions, Neptune generates multiple features. In general, if the chunks have one or more features (maximal values) in common, they are likely to be very similar. In the meantime, small changes to the data are unlikely to perturb the maximal values.

In practice, in order to generate multiple independent features, we use the Rabin fingerprint over rolling windows w of chunk C and compare the fingerprint FP against a mask. We then permute the Rabin fingerprint to generate multiple values with function β_i. The function leverages randomly generated multiplier with 32-byte windows and adder values m and a.

In general, if the maximal values are not changed, we can achieve a resemblance match. We hence group multiple features together to build a "super-feature." The super-feature value serves as a representation of the underlying feature values. It has the salient property that if two chunks have an identical super-feature, all the underlying features will match well. The super-features help identify the chunks that are similar for a match.

After we compute all features, a super-feature SF_j is built via a Rabin fingerprint over k consecutive features. We represent consecutive features as $\mathrm{feature}_{b\ldots e}$ for beginning and ending positions b and e, respectively.

In order to obtain a suitable tradeoff between the number of features and the super-feature's quality, we performed a large number of experiments. The experiments were completed with Campus datasets as described in Section IV-A. The Campus set contained multiple weeks of backups and had variable-size data from several to hundreds of Terabytes. We use the variable numbers of features per super-feature and the super-features per sketch. We observe that increasing the number of features per super-feature will increase the quality of matches but decrease the number of the identified matches. On the other hand, if we increase the number of super-features, the number of matches increases, unfortunately causing the increase in the indexing overheads. We typically identify that four features per super-feature can obtain the suitable tradeoff that exhibits good resemblance matches.

A resemblance lookup is executed in an index representing the corresponding super-features of previously processed chunks. We use each super-feature as a query request. Chunks that match on more super-features are considered better matches than those that match on fewer superfeatures, which is helpful to our approximate delta compression in the remote backups.

B. Data Filtration

Data filtration in Neptune consists of chunk-level deduplication and approximate delta compression.

First, the deduplication has become a key component in modern backup and archive systems. It eliminates duplicate data, thus effectively improving the system efficiency. The deduplication divides a data stream into variable-sized chunks (e.g., 8 kB on average), and then replaces duplicate chunks with pointers to their previously stored copies. A deduplication system identifies each chunk by its hash digest, namely fingerprint. A fingerprint index is used to map fingerprints of the stored chunks to their physical addresses. In practice, due to the variable sizes of chunks, a deduplication system manages data at a larger unit, called container. A container is fixed sized (e.g., 4 MB). A container is the basic unit of read and write operations. For a backup, Neptune aggregates the chunks into the containers to preserve the locality of the data stream. Moreover, for a restore, Neptune uses the container as the prefetching unit. A least recently used (LRU) algorithm is used to evict a container from the restore cache.

Second, the delta compression is designed as a faster and more efficient way. It leverages the similarities between files and can create significantly small compressed files. Hence, in order to improve network transmission, transmitting only the difference (or delta) between two files requires a significantly smaller number of bits. Delta compression uses a compressor that accepts two inputs. One is the target file to be compressed.

The other is a reference source file. The delta creator locates and copies the differences between the target and source files, compressing only those differences as a delta. The decompressor takes the delta and source files as the input to create an exact copy of the target.

C. Delta Chains

The time overheads in terms of transmitting files to and from a server are directly proportional to the amounts of data to be sent. For a delta backup and restore system, the amount of data is also related with the manner in which delta files are generated.

1) Linear Delta Chains: In storage backup systems, a linear sequence of versions constructs the history of modifications to a file. The sequence is generally called a version chain. The version chain may result in frequent access to old data and cause more transmission delays. Conventional schemes mainly use a linear version chain which is a compact version storage scheme. The linear chain can represent the interversion modification between consecutive versions.

We denote the uncompressed ith version of a file by V_i. The difference between two versions V_i and V_j is $\Delta_{(V_i,V_j)}$. The file $\Delta_{(V_i,V_j)}$ represents the differentially compressed encoding of V_j via V_i, which allows V_j to be restored by the inverse differencing operation in V_i and $\Delta_{(V_i,V_j)}$. The differencing operation is represented as $\delta(V_i, V_j) \to \Delta_{(V_i,V_j)}$. Moreover, we present the inverse differencing or reconstruction operation to be $\delta^{-1}(\Delta_{(V_i,V_j)}, V_i) \to V_j$. In the context of a linear-version chain, we say that versions V_i and V_j are adjacent if $j - i = 1$. We can build V_i via the modification of V_{i-1}.

For a file, the linear sequence of versions is $V_1, V_2, \ldots, V_{i-1}, V_i, V_{i+1}, \ldots$. In order to store and maintain these version, conventional approaches generally leverage a series of deltas. For example, two adjacent versions V_i and V_{i+1} are used to store the difference between these two files, $\Delta_{V_i,V_{i+1}}$. The adjacent versions hence lead to a "delta chain," such as $V_1, \Delta_{(V_1,V_2)}, \ldots, \Delta_{(V_{i-1},V_i)}, \Delta_{(V_i,V_{i+1})}, \ldots$. In the context of delta chains, we need to iteratively execute the inverse differencing algorithm on all intermediate versions, i.e., from 2 to i, to reconstruct the version V_i.

In terms of time overhead of rebuilding a version, we need to reconstruct all of the intermediate versions. In general, the reconstruction time is linear to the number of intermediate versions.

2) Shortcut Delta Chains: In order to improve the restore performance and reduce the operation delays, we propose a shortcut scheme for multiversion delta chains. The shortcut scheme leverages a minimum number of files for reconstruction. This delta chain consists of the modified forward deltas and a randomly selected file, $V_1, \Delta_{(V_1,V_2)}, \Delta_{(V_1,V_3)}, \ldots, \Delta_{(V_1,V_{i-1})}, V_i, \Delta_{(V_i,V_{i+1})}, \ldots$.

The shortcut chain has the benefit of allowing any given version to be reconstructed by accessing at most two files from the version chain. When a client executes the delta compression, the files to be transmitted to the server can be stored. Moreover, besides adjacent versions, Neptune needs to offer efficient and scalable delta compression for two nonadjacent versions, say, versions V_i and V_j. We use $|\Delta_{(V_i,V_j)}|$ to represent the size of $\Delta_{(V_i,V_j)}$. If $j - i$ increases, the size will increase, thus, leading to potential decrease of the compression quality.

In general, two adjacent versions V_i and V_{i+1} have $\alpha|V_i|$-modified fractions between them. The parameter α represents the compression quality between adjacent versions. Neptune makes use of a differencing algorithm to build a delta file $\Delta_{(V_i,V_{i+1})}$. The larger the size $\alpha|V_i|$ is, the better the compression performance is. The version compression is given by $V_i = 1 - \frac{|\Delta_{(V_i,V_{i+1})}|}{|V_{i+1}|}$.

This result mainly demonstrates the relative compressibility of all new versions that have the identical-size deltas. However, for the worst-case compression, the size of the delta file is $\alpha|V_i|$, and the new version has the variable sizes from $|V_i|$ to $(1 + \alpha)|V_i|$. The $\alpha|V_i|$-modified fractions in V_{i+1} are able to replace existing fractions in V_i, i.e., $|V_i| = |V_{i+1}|$. We further analyze the worst-case size of shortcut delta chains. Specifically, the maximum size of the modified fractions is $\alpha|V_i|$ between versions V_i and V_{i+1}. Between versions V_{i+1} and V_{i+2}, we can obtain at most $\alpha|V_{i+1}|$-modified fractions. Hence, between versions V_i and V_{i+2}, by carrying out the union bound on the number of the modified fractions, we obtain at most $2\alpha|V_i|$ modified fractions.

D. Delta Compression and Recovery

Compression techniques are widely used in computer networks and storage systems to increase the efficiency of data transmission and reduce space requirements on the end systems. Delta encoding greatly reduces data redundancy. Collections of unique deltas are substantially more space efficient than their nonencoded equivalents. In practice, there are two inputs, i.e., target file to be compressed and a reference source file. Encoding deltas need to compress the difference between the target and source file as a delta. On the other hand, decoding delta takes the delta and source files as inputs to generate an exact copy of the target.

The rationale of delta compression comes from the fact that both sender and receiver contain a reference file that is similar to the transmitted file. Hence, we only need to transmit the difference (or delta) between the two files, which requires a significantly smaller number of bits. Formally, we have two files $f_{\text{new}}, f_{\text{old}}$, and a client C and a server S connected by a communication link. C has a copy of f_{new}, and S has a copy of f_{old}. The design goal is to compute a file f_δ of minimum size, such that S can reconstruct f_{new} from f_{old} and f_δ. f_δ is called as a delta of f_{new} and f_{old}.

The implementation details are described as follows. Delta compression can compress files by calculating the differences of similar regions relative to some reference/base region. Once a base file is selected for delta compression, it can be divided into multiple base chunks. To perform delta encoding, we use a technique based on Xdelta [8] which is optimized for compressing highly similar data regions. Specifically, we initialize the encoding by iterating through the base chunk, calculating a hash value at subsampled positions, and storing the hash and offset in a temporary index. We then begin processing the target chunk from a target file by calculating a hash value at rolling

TABLE I
DATA PROPERTIES IN TERMS OF FILE TYPES FOR CAMPUS DATA

	iso	rar/zip	exe	gif/jpeg/bmp	htm	c/h/lib	wav/mp3	dll/sys/int	pdf	txt/doc/xls	avi/rmvb/mp4	others
File Numbers (million)	0.087	2.173	0.591	952.618	142.953	3.819	328.472	0.849	12.739	17.485	2.681	0.854
Total size (TB)	9.6	33.2	24.8	528.1	94.5	25.5	142.5	1.2	114.7	123.2	303.9	91.5
Capacity percentage (%)	0.6	2.2	1.6	35.2	6.3	1.7	9.5	0.8	7.6	8.2	20.2	6.1
Modified files in a period (%)	0.026	0.103	0.227	15.748	0.042	6.871	0.042	0.847	37.816	69.628	6.415	5.942
Exact-matching duplicate (%)	7.9	10.3	2.8	26.3	12.4	9.2	10.5	6.8	20.4	16.9	4.2	7.3

window positions. We look up the hash value in the index to find a match against the base chunk. If there is a match, we compare bytes in the base and target chunks forward and backward from the starting position to create the longest match possible. If the bytes fail to match, we insert the target's bytes into the output buffer, and we also add this region to the hash index.

In general, users may choose to recover some files at some points after their deletion. Although the design goal of Neptune is to remove near duplicate files, the recovery function is useful in improving system usability and allows users to flexibly and easily use the Neptune scheme. In order to support the recovery functionality, a simple but naive solution is to maintain the deleted files as backups, which consumes substantial system resources (computation and space overheads).

In order to offer a cost-effective and efficient recovery solution, we propose a delta-based decompression scheme that computes the difference among multiple similar files against a base one. The base file can be artificially selected by users or automatically by Neptune that leverages well-recognized clustering algorithms. Neptune only needs to maintain the base file (not deduplicated) and the deltas from other deduplicated files. The delta-based design can significantly reduce the demand for system resources while supporting the recovery functionality. In order to facilitate the delta decoding in the delta recovery, Neptune maintains all files that were ever base ones, even if they are selected to be removed by some other users. When users want to recover files, Neptune executes the delta decoding operations by computing the base files and the deltas.

IV. PERFORMANCE EVALUATION

A. Experiments Setup

We have implemented Neptune between two data centers, and their distance is more than 1200 km via 2 Mb/s network link. Each center consists of 128 servers and each server has a 8-core CPU, a 32-GB RAM, a 500-GB 7200 r/min hard disk, and Gigabit network interface card. The Neptune prototype implementation required approximately 6000 lines of C code in a Linux environment.

To examine the system performance, we leverage three typical backup datasets, i.e., LANL [20], EMC [21], and a Campus collection from real-world cloud backup applications. We describe the characteristics of the Campus dataset.

1) Campus: In order to comprehensively examine the performance, we use a real collection from a campus cloud system. This system offers cloud storage services for more than 70 000 users, including faculty, staff, graduate students (Ph.D. and Master), and undergraduate students. Each user is allowed to use 50-GB capacity, and the entire used storage capacity is more than 1.5 PB. As shown in Table I, we report the properties of the stored data after average 2-year use. We also examine the percentage of the modified files in each type. Moreover, the office and pdf files are frequently modified. Performing the delta compression upon them can obtain more bandwidth savings. More than 15% files have been modified, meaning that they can also be delta compressed within backup periods. Furthermore, by leveraging the exact matching chunk-level detection, we can identify duplicates of file, pdf, and office files. In addition, the near-duplicate files occupy more than 57.2% percentage of entire dataset.

Neptune can support top-k approximate delta compression and deliver high system performance. The k value determines the number of transmitted fingerprints from destination to source servers to identify similar fingerprints. In general, the larger the k value is, the more the base fingerprints can be found, which unfortunately incur the longer latency due to computation-intensive indexing. In order to select suitable k values, we attempt to examine the tradeoff between obtained bandwidth savings and execution latency. The used metric is the normalized rate that is the value of the saved bandwidth divided by the indexing latency. Specifically, we count the bandwidth against different k values and compute their normalized values between the minimum and maximum values. In the similar way, we compute the normalized latency values of executing top-k indexing. We observe that the selected k values are respectively 4, 5, and 6 for three used sets. These k values can obtain suitable tradeoff between bandwidth savings and indexing latency.

To the best of our knowledge, there are no existing approaches that can support both local deduplication and delta compression for remote communications. In order to carry out fair and meaningful comparisons, we, respectively, compare Neptune with these two aspects. For deduplication, we compare Neptune with state-of-the-art deduplication schemes, including EndRE [2] and CBD [3]. For delta compression, we compare Neptune with EMC SIDC [4] that is a salient feature of backup replication in EMC backup recovery systems. Moreover, due to no open source codes, we choose to reimplement EndRE [2], CBD [3], and SIDC [4]. Specifically, we implemented EndRE's end-host-based redundancy elimination service, which includes adaptive algorithm (i.e., SampleByte) and its optimized data structure. We also implemented the components of CBD, including fingerprint cache, containers, and super-chunk-based data routing scheme. SIDC was implemented, including its Bloom filter, fingerprint index, and containers, to load the stored sketches into a stream-informed cache.

B. Results and Analysis

1) Deduplication Throughput: Fig. 1 shows the results in terms of deduplication throughput. Specifically, Neptune achieves an average throughput of about 3.36 GB/s, higher

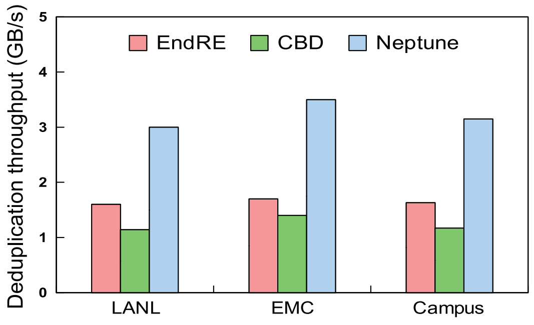

Fig. 1. Deduplication throughput.

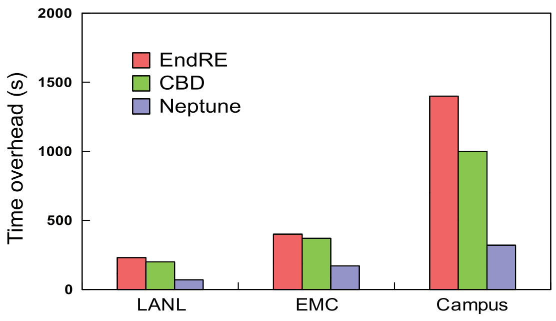

Fig. 2. Time overhead in the file deduplication.

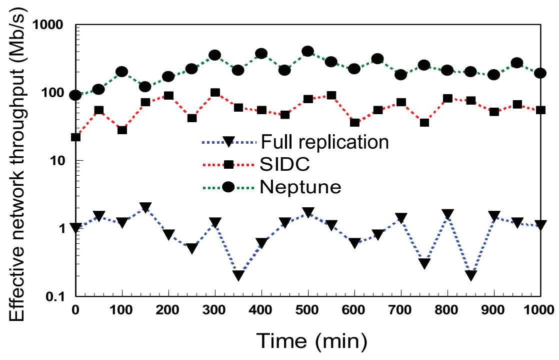

Fig. 3. Effective throughput in remote communications.

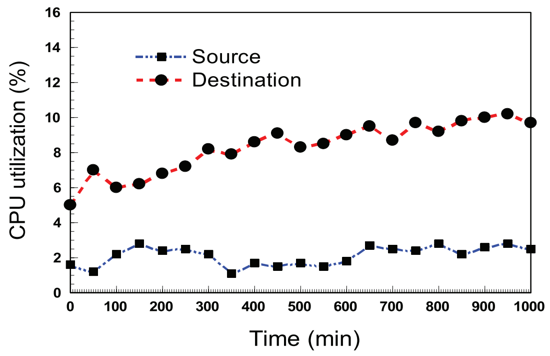

Fig. 4 Delta computation overheads.

than 1.92 GB/s in EndRE and 1.35 GB/s in CBD. CBD has to consume more memory to handle stateful routing with high overhead, thus, obtaining relatively low performance. The substantial throughput improvements of Neptune attribute to the LSH-based grouping scheme that can significantly narrow the scope of processing data while alleviating the overheads and improving the deduplication throughput. In general, EndRE can deliver high performance when relatively large memory is available. However, in the context of this paper, the memory capacity is limited in remote backups. Although EndRE optimizes data structures to reduce memory overhead, its fingerprinting hash table in practice consumes substantial space that is much larger than the limited memory size. EndRE hence has to frequently access to the noncached fingerprints in hard disks, thus decreasing the throughput.

2) Time Overhead: We examine the time overhead in completing the file deduplication and the experimental results are shown in Fig. 2. EndRE leverages sample-based fingerprinting algorithm to accelerate the deduplication. Neptune obtains the smallest time overhead due to the usage of the LSH computation to implement the fast and accurate detection of duplicates.

3) Effective Network Throughput: We perform numerous remote communications' experiments to measure effective network throughput between two remote cities. Their distance is more than 1200 km via 2 Mb/s network link. Fig. 3 shows the results of the Campus dataset that has the largest size. The throughput runs at 2 Mb/s and is measured every 20 min. We observe that compared with full replication (i.e., the baseline), the average effective throughputs in Neptune and SIDC are 286.27 and 57.26 Mb/s, respectively, much larger than the baseline 1.57 Mb/s. The main reason is that both Neptune and SIDC leverage the delta compression that physically transfers much less data across the network. Furthermore, SIDC replies on the closest fingerprint to determine the similarity. Performing the delta compression on a single base chunk limits the utilization and causes the decrease of communication performance. The reason is that SIDC often fails to identify similar chunks to be compressed. Unlike it, Neptune obtains better performance by finding more chunks that can be delta compressed.

4) Delta Computation Overheads: Neptune leverages the delta compression to improve the effective throughput, which meantime incurs extra computation and disk I/O overheads. We examine these overheads in local servers and remote backup servers. First, the storage capacity overheads for maintaining fingerprints are relatively small. Each chunk stored in a container (after deduplication) has the fingerprints. The fingerprints are added to the metadata section of the storage container, which is less than 40 bytes. The disk I/O overhead is modest (around 4.5%). Furthermore, since the main overhead comes from the computation cost, we examine the real CPU utilization in both source and destination servers, as shown in Fig. 4.

We measure the CPU utilization over every 2-min period after the initial seeding phase. In the source servers, the CPU utilization is around 2.47% to mainly identify top-k fingerprints and compute the deltas to be transmitted. We also observe that in the destination server, the CPU utilization demonstrates increasing trend, from 4.7% to 9.8%. The main reason is that the CPU overhead, i.e., indexing upon Bloom filters and fingerprint structure, almost scales linearly as the number of transmitted data. Overall, the entire CPU overheads are no more than 10%, which is acceptable in the servers, due to the increasing

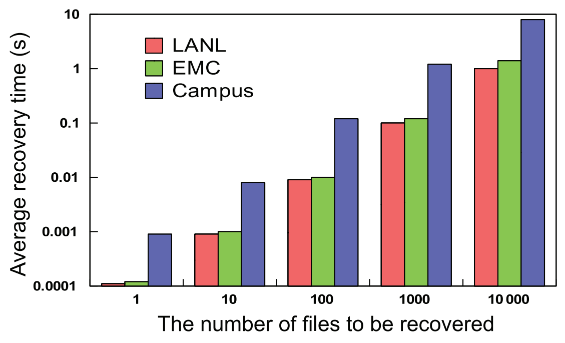

Fig. 5. Average recovery time.

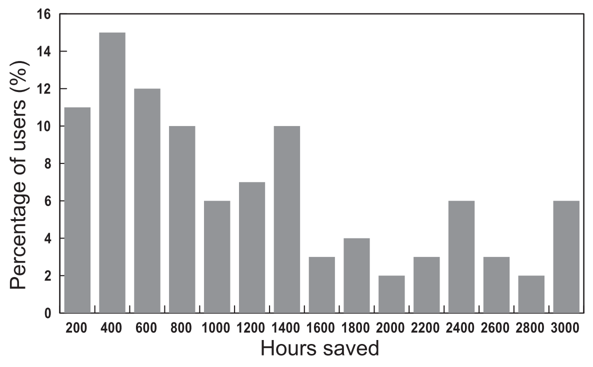

Fig. 6. Distribution of hours saved by users.

number of CPU cores. In essence, we trade computation and I/O resources for higher network throughput between remote network servers.

5) Recovery Overheads: The delta-based recovery scheme can remotely restore the removed files with relatively small space and computation overheads. Compared with full backups, the proposed delta-based recovery scheme in Neptune can significantly reduce the system overheads. Note that since most encoding operations can be completed in an offline manner, we mainly examine the decoding (i.e., recovery) time that is the main concern for users.

Fig. 5 shows the recovery time by examining the operation of decoding the deltas. We observe that the recovery time is approximately linear to the number of files to be recovered. Recovering 1000 files requires about 1 s, which is generally acceptable to users. As described in Section III-C2, Neptune leverages the shortcut chains to allow any given version to be restored by accessing at most two files from the version chain. Neptune hence avoids extra computation latency on the intermediate deltas and supports fast recovery.

6) Neptune Assessment by Users: We analyzed daily reports from cloud systems used by users during the second week of October 2012. Fig. 6 shows how much time was saved by the users versus sending data without any compression. We reported the amounts of the transmitted data, network throughput, and compression performance. We hence can calculate how long remote backups would take without compression. The median users would require 1025 h to fully replicate their data (more than 6 weeks). With the aid of Neptune, the backups were reduced to 18.5 h (saving over 1000 h of network transfer time).

TABLE II
USER FEEDBACKS FROM NEPTUNE (AVERAGE SCORES)

	Faculty	Staff	Graduate	Undergraduate	Average
Usability	4.2	4.7	4.4	4.7	4.5
Clarity	4.0	4.3	4.1	4.4	4.2
Completeness	3.9	4.2	4.1	3.8	4.0
Reliability	4.2	4.3	4.0	3.9	4.1
Portability	4.2	4.1	4.4	4.5	4.3
Consistency	4.0	4.1	3.9	4.4	4.1
Maintainability	4.1	4.3	3.9	3.7	4.0

TABLE III
USER FEEDBACKS FROM BASELINE (AVERAGE SCORES)

	Faculty	Staff	Graduate	Undergraduate	Average
Usability	3.3	3.5	3.2	3.6	3.4
Clarity	3.7	3.1	3.2	3.1	3.3
Completeness	3.1	3.2	3.0	3.1	3.1
Reliability	3.2	3.1	2.9	3.4	3.2
Portability	3.2	3.1	3.0	3.2	3.1
Consistency	3.0	3.1	3.4	3.2	3.2
Maintainability	2.7	3.0	3.1	3.2	3.0

We argue that it is important and intuitive to evaluate new backup service by considering users' experiences after they use a new system design. Users' feedbacks often show some important aspects that may not be revealed by either simulations or implementations, and thus serve to complement the prototype-based evaluations.

A total of 500 users, including 50 faculty members, 100 staff members, 150 graduate students, and 200 undergraduate students, were requested to use and evaluate a prototype version of Neptune after they use it. Users installed the client program of Neptune in their local operation systems (Windows or Linux). Neptune then allows the users to use the remote backups. Moreover, we also compare the feedbacks of Neptune with those of a baseline approach. The baseline approach can support backup service via direct transmission without deduplication and delta compression. Their experiences are rated on a scale of 1–5, i.e., 5 (Outstanding), 4 (Good), 3 (Satisfactory), 2 (Poor), and 1 (Unsatisfactory). The users' feedbacks are summarized in Tables II and III, respectively, for Neptune and Baseline.

The average evaluation score of the overall performance in Neptune is 4.17, which is much higher than 3.19 in Baseline. The results demonstrate that Neptune is satisfactory with respect to its effectiveness and efficiency. The usage of our deduplication scheme is proven to be easy.

V. CONCLUSION

In order to offer efficient remote communications in cloud backups, this paper proposed a cost-effective backup framework, called Neptune. Neptune is designed for cloud backup services by leveraging comprehensive data filtration, including local deduplication and network delta compression. To improve the network transmission performance, Neptune uses approximate delta compression to identify more chunks to be compressed. The approximate methodology can significantly decrease the complexity in terms of data compression and reduce the data to be transmitted. Moreover, the shortcut-chain approach leverages a minimum number of files for reconstruction and improves the compression performance by alleviating the computation overhead in the recovery. Neptune has been implemented and thoroughly evaluated in remote backup systems. Extensive experimental results demonstrate the benefits over state-of-the-art schemes including EndRE, CBD, and SIDC. In the future work, by exploiting and exploring the features of data filtration, we plan to use the proposed Neptune scheme into the real cloud systems and support efficient backup services.

References

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Symp. Theory Comput. (STOC)*, 1998, pp. 604–613.

[2] B. Aggarwal *et al.*, "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. Symp. Netw. Syst. Des. Implement. (NSDI)*, 2010, pp. 419–432.

[3] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2011, pp. 15–30.

[4] P. Shilane, M. Huang, G. Wallace, and W. Hsu, "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2012, pp. 49–64.

[5] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. Conf. File Storage Technol. (FAST)*, 2008, pp. 269–282.

[6] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th Conf. File Storage Technol. (FAST)*, 2009, pp. 111–123.

[7] D. Trendafilov, N. Memon, and T. Suel, "ZDelta: An efficient delta compression tool," CIS Dept., Polytechnic Univ., Lakeland, FL, USA, Tech. Rep. TR-CIS-2002-02, 2002.

[8] J. MacDonald, "File system support for delta compression," M.S. thesis, Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, 2000.

[9] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[10] J. J. Hunt, K.-P. Vo, and W. F. Tichy, "Delta algorithms: An empirical analysis," *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, vol. 7, no. 2, pp. 192–214, 1998.

[11] Z. Li *et al.*, "Towards network-level efficiency for cloud storage services," in *Proc. ACM Internet Meas. Conf. (IMC)*, 2014, pp. 115–128.

[12] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2004, pp. 59–72.

[13] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered approach for eliminating redundancy in replica synchronization," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2005, pp. 281–294.

[14] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2010, pp. 17–32.

[15] V. Ramasubramanian *et al.*, "Cimbiosys: A platform for content-based partial replication," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2009, pp. 261–276.

[16] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proc. SIGCOMM Conf.*, 2012, pp. 431–442.

[17] Z. Li *et al.*, "Efficient batched synchronization in dropbox-like cloud storage services," in *Proc. Middleware*, 2013, pp. 307–327.

[18] Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient remote communication services for cloud backups," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2014, pp. 844–852.

[19] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, "Design and implementation of a pipelined datapath for high-speed face detection using FPGA," *IEEE Trans. Ind. Inf.*, vol. 8, no. 1, pp. 158–167, Feb. 2012.

[20] Los Alamos National Lab (LANL). *File System Data* [Online]. Available: http://institute.lanl.gov/data/archive-data/

[21] EMC. (2014). *Backup File-system Snapshots* [Online]. Available: http://tracer.filesystems.org/s

Yu Hua (SM'13) received the B.E. and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 2001 and 2005, respectively.

He is an Associate Professor with the Huazhong University of Science and Technology, Wuhan. He has authored or co-authored more than 60 papers published in major journals and international conference proceedings including USENIX Annual Technical Conference (USENIX ATC), File and Storage Technologies (FAST), IEEE International Conference on Computer Communications (INFOCOM), and International Conference for High Performance Computing, Networking, Storage and Analysis (SC). His research interests include computer architecture, cloud computing, and network storage.

Xue Liu (M'06) received the B.S. degree in mathematics and the M.S. degree in automatic control from Tsinghua University, Beijing, China, 1996 and 1999, respectively and the Ph.D. degree in computer science from the University of Illinois, Urbana–Champaign, Champaign, IL, USA, in 2006.

He is an Associate Professor with the School of Computer Science, McGill University, Montreal, QC, Canada. His research interests include computer networks and communications, smart grid, real-time and embedded systems, and cyber–physical systems.

Dr. Liu is a member of ACM.

Dan Feng (M'05) received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively.

She is a Professor and Vice Dean with the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems.