ISSN:2348-2079

Volume-7 Issue-1

International Journal of Intellectual Advancements and Research in Engineering Computations

Pack: prediction-based cloud bandwidth and cost decrement system

Mr.C.Mani MCA., M.Phil., ME[1], Mr. K. Siragirivelavan[2]

[1]Associate Professor, Nandha Engineering College (Autonomous), Erode-52.

[2]MCA Nandha Engineering College (Autonomous), Erode-52.

ABSTRACT

In this paper, we present PACK (Predictive ACKs), a novel start to finish traffic excess disposal (TRE) framework, intended for distributed computing clients. Cloud-based TRE needs to apply a sensible utilization of cloud assets so the transfer speed cost Decrement joined with the extra expense of TRE calculation and capacity would be improved. PACK's primary leeway is its ability of offloading the cloud-server TRE exertion to endclients, in this manner limiting the preparing costs incited by the TRE calculation. In contrast to past arrangements, PACK does not require the server to persistently keep up customers' status. This makes PACK truly reasonable for inescapable calculation conditions that consolidate customer versatility and server movement to keep up cloud flexibility. PACK depends on a novel TRE method, which enables the customer to utilize recently gotten lumps to distinguish recently gotten piece chains, which thus can be utilized as solid indicators to future transmitted lumps.

Index Terms: Cloud Computing, Predictive Acks, Network optimization, Secure Hash Algorithm-1, Advanced Encryption standard, Traffic Redundancy Elimination.

INTRODUCTION

Network traffic exhibits large amount of redundancy when different users on the Internet access same or similar content. TRE is used to eliminate the transmission of redundant content and, therefore, to significantly reduce the network cost. In most common TRE solutions, both the sender and the receiver examine and compare signatures of data chunks, parsed according to the data content prior to their transmission. When redundant chunks are detected, the sender replaces the transmission of each redundant chunk with its strong signature. Commercial TRE solutions are popular at enterprise networks, and involve the deployment of two or more proprietary protocol, state synchronized middle-boxes at both the intranet entry points of data centers and branch offices, eliminating repetitive traffic between them. Example: Riverbed solution which will compress the data before sending [1-3].

While proprietary middle-boxes are popular point solutions within enterprises, they are not as attractive in a cloud environment. Current end-to-end TRE solutions are sender-based. In the case where the cloud server is the sender, these solutions require that the server continuously maintain clients' status. Cloud elasticity calls for a new TRE solution. The popularity of rich media that consume high bandwidth.

Motivates content distribution network (CDN) solutions, in which the service point for fixed and mobile users may change dynamically according to the relative service point locations and loads.

Our new traffic redundancy elimination approach also called PACK (Predictive ACKs) or a novel end-to-end traffic redundancy elimination (TRE) system, which detects redundancy at the client side and there is no need of server to continuously. The basic unit of data is the chunk. The receiver implements a chunk store which

Author for correspondence:
Department of MCA, Nandha College of Technology, Erode – 638052, Tamil Nadu, India

operates by the LRU (Line Replacement Unit) principle. Chunks are recognized by "hints" which are easily computed functions which identify data with a fast algorithm with low false positives. If hints match then SHA-1 signatures are checked and if matched then data about successive chunks (referred to as a "chain") can be got back from pointers in chunk data. So, a chunk is the data, a signature to uniquely identify the data and a link to the next chunk (if received) [4].

On receiving a match the receiver asks the sender if the next chunks will be the same as the ones in its store by sending the identity of several chunks. This is known as a PRED command (predict). If this is correct the sender sends PRED-ACK and the receiver sends the chunks to its own TCP input buffers. If the prediction is not correct the sender continues to send as normal. The TCP Options field is used to carry the PACK wired protocol. In order to get optimal resource utilization and decrease computing time load balancing is implemented in this system. This also helps to increase the capacity of a server farm beyond that of a single server [5].

RELATED WORK

There has been an enormous and steady usage of several TRE techniques. In this paper a technique for identifying repetitive information transfers and use it to analyze the redundancy of network traffic has been suggested. Here the insight is that dynamic content, streaming, media and other traffic that is not caught by today's web caches is nonetheless likely to derive from similar information. Hence similarity detection techniques have been adopted for designing a system to eliminate redundant transfers. And to identify repeated byte ranges between packets to avoid retransmitting the redundant data. Users rarely consider running network file systems over slow or wide-area networks, as the performance would be unacceptable and the bandwidth consumption is too high [6-9].

This paper presents LBFS (low- bandwidth network file system), a network file system designed for low-bandwidth networks. LBFS exploits similarities between files or versions of the same file to save bandwidth. It avoids sending data over the network when the same data can already be found in the server's file system or the client's cache. Using this technique in conjunction with conventional compression and caching, LBFS consumes over an order of magnitude less bandwidth than traditional network file systems on common workloads.

In this paper key idea in the prediction algorithm is to treat a set of previously observed traffic matrices as "experts" and learn online the best weighted linear combination of these experts to make its prediction. With tenant VM placement using these predictive guarantees, we find that the inter-rack network utilization in certain datacenter topologies can be more than doubled.

In this paper based on apparatus for reducing network traffic. At sender a data chunk is identified for transmission to a receiver, which is connected to the sender over a communication link. The sender computes a signature of the data chunk and determines whether the data chunk has been previously transmitted by looking up the signature in a sender index table. The sender index table associates the signatures of previously transmitted data chunks with unique index values. A message is transmitted to the receiver, where if the data chunk has previously been transmitted then the message includes an index value from the sender index table that is associated with the signature of the data chunk. At the receiver, the data chunk is located in a receiver cache that stores the previously transmitted data chunks by looking up the index value included in the message in a receiver index table. The receiver index table associates the unique index values with the locations in the receiver cache of the previously transmitted data chunks.

With the advent of the cloud computing, Developers with innovative ideas for new internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it. They need not be concerned about over provisioning for a service whose popularity does not meet their predictions, thus wasting costly resources, or under provisioning for one that becomes wildly popular, thus missing potential customers and revenue.

In last couple of years there huge increase in the usage cloud computing because cloud

computing is emerging style of IT-delivery in which applications, data and resources are rapidly provisioned provided as Standardized offerings to users with a flexible price. But it is important to provide the convenient pricing model for the users of cloud. Hence, here a new traffic redundancy and elimination scheme has been designed for reducing the cloud bandwidth and costs.

It is a software-based middle-box replacement for the expensive commercial hardware. In this scheme, the sender middle-box holds back the TCP stream and sends data signatures to the receiver middle-box. The receiver checks whether the data is found in its local cache. Data chunks that are not found in the cache are fetched from the sender middle-box or a nearby receiver middle-box.

It uses a new chunking scheme that is faster than the commonly used Rabin fingerprint, but is restricted to chunks as small as 32–64 B. Unlike PACK, End RE requires the server to maintain a fully and reliably synchronized cache for each client.

To adhere with the server's memory requirements, these caches are kept small (around 10 MB per client), making the system inadequate for medium-to-large content or long-term redundancy. End RE is server-specific, hence not suitable for a CDN or cloud environment.

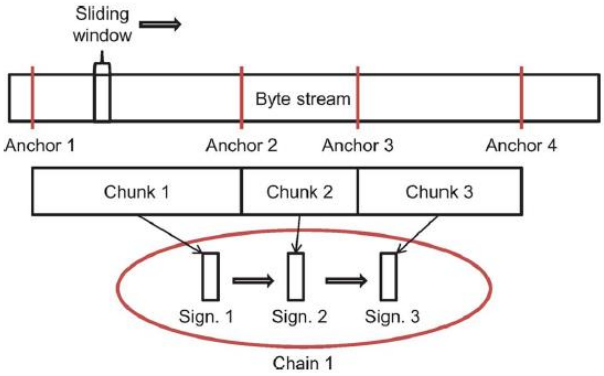

Fig. 1. From stream to chain.

PACK ALGORITHM

The chunks are then compared to the receiver local storage, termed chunk store. If a matching chunk is found in the local chunk store, the receiver retrieves the sequence of subsequent chunks, referred to as a chain, by traversing the sequence of LRU chunk pointers that are included in the chunks' metadata. Using the constructed chain, the receiver sends a prediction to the sender for the subsequent data. Part of each chunk's prediction, termed a hint, is an easy to compute function with a small enough false-positive value, such as the value of the last byte in the predicted data or a byte-wide XOR checksum of all or selected bytes.

The sender distinguishes the anticipated range in its cushioned information and confirms the indication for that go. On the off chance that the outcome coordinates the got clue, it keeps on playing out the more computationally concentrated SHA-1 signature task. Upon a mark coordinate, the sender sends an affirmation message to the collector, empowering it to duplicate the coordinated information from its neighborhood stockpiling.

Receiver Chunk Store

PACK uses a new *chains* scheme, which chunks are linked to other chunks according to their last Received order. The PACK receiver maintains a *chunk store*, which is a large size cache of chunks and their associated metadata. Chunk's metadata includes the chunk's signature and a (single) pointer to the successive chunk in the last received stream containing this chunk. Caching and indexing techniques are employed to efficiently maintain and retrieve the stored chunks,

their signatures, and the chains formed by traversing the chunk pointers.

When the new data are received and parsed to chunks, the Receiver computes each chunk's signature using SHA-1. At this point, the chunk and its signature are added to the chunk store. In addition, the metadata of the previously received chunk in the same stream is updated to point to the current chunk. The unsynchronized nature of PACK allows the receiver to map each existing file in the local file system to a chain of chunks, saving in the chunk store only the metadata associated with the chunks. Using the latter observation, the receiver can also share chunks with peer clients within the same local network utilizing a simple map of network drives.

The utilization of a small chunk size presents better redundancy Elimination when data modifications are fine-grained, such as sporadic changes in an HTML page. On the other hand, the use of smaller chunks increases the storage index size, memory usage, and magnetic disk seeks. It also increases the transmission overhead of the virtual data exchanged between the client and the server.

Unlike IP-level TRE solutions that are limited by the IP

Packet size (B), PACK operates on TCP streams and can therefore handle large chunks and entire chains. Although our design permits each PACK client to use any chunk size, we recommend an average chunk size of 8 KB.

Receiver Algorithm

Upon the arrival of new data, the receiver computes the respective signature for each chunk and looks for a match in its local chunk store. If the chunk's signature is found, the receiver determines whether it is a part of a formerly received chain, using the chunks' metadata. If affirmative, the receiver sends a prediction to the sender for several next expected chain chunks.

The prediction carries a starting point in the byte stream (i.e.,offset) and the identity of several subsequent chunks (PRED command).

Upon a fruitful forecast, the sender reacts with a PRED-ACK affirmation message. When the PRED-ACK message is gotten and handled, the recipient duplicates the relating information from the lump store to its TCP input cradles, putting it as indicated by the comparing arrangement numbers. Now, the beneficiary sends a typical TCP ACK with the following expected TCP succession number. On the off chance that the forecast is false, or at least one anticipated lumps are as of now sent, the sender proceeds with ordinary task, e.g., sending the crude information, without sending a PRED-ACK message.

Proc. 1: Receiver Segment Processing

1. **if** segment carries payload *data* **then**
2. calculate chunk
3. **if** reached chunk boundary **then**
4. activate predAttempt()
5. **end if**
6. **else if** PRED-ACK segment **then**
7. processPredAck()
8. activate predAttempt()
9. **end if**

Proc. 2: predAttempt()

1. **if** received *chunk* matches one in chunk store **then**
2. **if** foundChain(*chunk*) **then**
3. prepare PREDs
4. send single TCP ACK with PREDs according to Options free space
5. exit
6. **end if**
7. **else**
8. store *chunk*
9. link *chunk* to current chain
10. **end if**
11. send TCP ACK only

Proc. 3: processPredAck()

1. **for all** offset PRED-ACK **do**
2. read data from chunk store
3. put data in TCP input buffer
4. **end for**

Sender Algorithm

When a sender receives a PRED message from the receiver, it tries to match the received predictions to its buffered (yet to be sent) data. For each prediction, the sender determines the corresponding TCP sequence range and verifies the

hint. Upon a hint match, the sender calculates the more computationally intensive SHA-1 signature for the predicted data range and compares the result to the signature received in the PRED message. Note that in case the hint does not match, a computationally expansive operation is saved. If the two SHA-1 signatures match, the sender can safely assume that the receiver's prediction is correct. In this case, it replaces the corresponding outgoing buffered data with a PRED-ACK message.

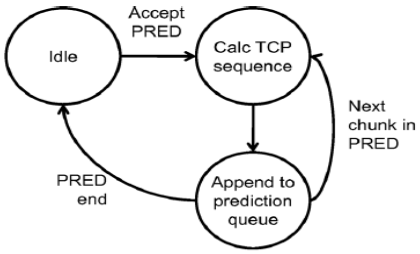

Fig. 2(a) filling the prediction queue

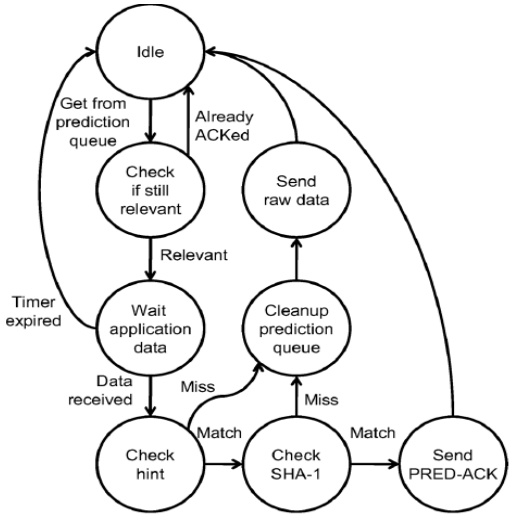

Fig. 2(b) Processing the prediction queue and sending PRED-ACK or raw data.

The receiver, in turn, triggers a TCP ACK message and Includes the prediction in the packet's Options field.

OPTIMIZATIONS

Adaptive Receiver Virtual Window

Predictive ACK enable the receiver side to locally capture the sender data when a local or temporary copy is available, thus eliminating the requirement to send this information through the network. In this term the receiver's fetching of that recent local data as the reception of visual data.

Cloud Server Acting as a Receiver

In a developing trend, cloud computing storage is getting a dominant player from backup of store and sharing of data services to the American National Library and e –mail services. In this most of these Services, the cloud is used often the receiver of the data.

Hybrid Approach

PACK's receiver-based mode is less efficient if changes in the data are scattered. In this case, the prediction sequences are frequently interrupted,

which, in turn, forces the sender to revert to raw data transmission until a new match is found at the receiver and reported back to the sender.

MOTIVATING A RECEIVER-BASED APPROACH

The objective of this section is twofold: evaluating the potential data redundancy for several applications that are likely to reside in a cloud, and to estimate the PACK performance and cloud costs of the redundancy elimination process.

Our evaluations are conducted using: 1) Video traces captured at a major ISP; 2) Traffic obtained from a popular social network service; and 3) Genuine data sets of real-life workloads. In this section, we relate to an average chunk size of 8 KB, although our algorithm allows each client to use a different chunk size.

TRAFFIC REDUNDANCY

Traffic Traces

We obtained a 24-h recording of traffic at an ISP's 10-Gb/s PoP router, using a 2.4-GHz CPU recording machine with 2 TB storage (4 500 GB 7 200 RPM disks) and 1-Gb/s NIC. We filtered YouTube traffic using deep packet inspection and mirrored traffic associated with YouTube servers IP addresses to our recording device. Our measurements show that YouTube traffic accounts for 13% of the total daily Web traffic volume of this ISP. The recording of the full YouTube stream would require 3 times our network and disk write speeds. Therefore, we isolated 1/6 of the obtained YouTube traffic, grouped by the video identifier (keeping the redundancy level intact) using a programmed load balancer that examined the upstream HTTP requests and redirected downstream sessions according to the video identifier that was found in the YouTube's URLs, to a total of 1.55 TB. For accurate reading of the true redundancy, we filtered out the client IP addresses that were used too intensively to represent a single user and were assumed to represent a NAT address. Note that YouTube's video content is not cacheable by standard Web proxies since its URL contains private single-use tokens changed with each HTTP request. Moreover, most Web browsers cannot cache and reuse partial movie downloads that occur when end-users skip within a movie or switch to another movie before the previous one ends. Table I summarizes our findings. We recorded more than 146 K distinct sessions, in which 37 K users request over 39 K distinct movies. Average movie size is 15 MB, while the

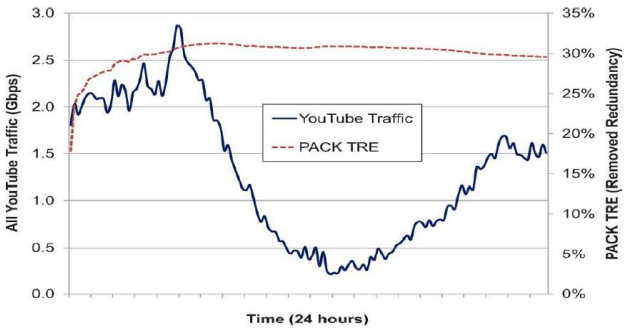

Fig.3. ISP's YouTube traffic over 24 h, and PACK redundancy elimination ratio with this data.

Average session size is 12 MB, with the difference stemming from end-user skips and interrupts. When the data is sliced into 8-kB chunks, PACK brings a traffic savings of up to 30%, assuming the end-users start with an empty cache, which is a worst-case scenario. Fig.3 presents the YouTube traffic and the redundancy obtained by PACK over the entire period, with the redundancy sampled every 10 min and averaged. This end-to-end redundancy arises solely from

self-similarity in the traffic created by end-users. We further analyzed these cases and found that end-users very often download the same movie or parts of it repeatedly. The latter is mainly an intersession redundancy produced by end-users that skip forward and backward in a movie and producing several (partially) overlapping downloads. Such skips occurred at 15% of the sessions and mostly in long movies (over 50 MB). Since we assume the cache is empty at the beginning, it takes a while for the chunk cache to fill up and enter a steady state. In the steady state, around 30% of the traffic is identified as redundant and removed. We explain the length of the warm-up time by the fact that YouTube allows browsers to cache movies for 4 h, which results in some replays that do not produce downloads at all.

Static Dataset

We acquired the following static datasets

Linux source—different Linux kernel versions: all the 40 2.0.x tar files of the kernel source code that sum up to 1 GB;

Email—a single-user Gmail account with 1140 e-mail messages over a year that sum up to 1.09 GB. The 40 Linux source versions were released over a period of 2 years. All tar files in the original release order, from 2.0.1 to 2.0.40, were downloaded to a download directory, mapped by PACK, to measure the amount of redundancy in the resulted traffic. Altogether, the Linux source files show 83.1% redundancy, which accounts to 830 MB. The total measured traffic redundancy was 31.6%, which is roughly 350 MB.

For example, a Gmail user that reads the same attachment for 10 times, directly from the Web browser, generates 90% redundant traffic. Our experiments show that in order to derive an efficient PACK redundancy elimination, the chunk-level redundancy needs to be applied along long chains. To quantify this phenomenon,

We explored the distribution of redundant chains in the Linux and Email datasets. Linux source presents the resulted redundant data chain length distribution. In Linux, 54% of the chunks are found in chains, and in Email about 88%.Moreover, redundant chunks are more probable to reside in long chains. These findings sustain our conclusion that once redundancy is discovered in a single chunk, it is likely to continue in subsequent chunks. Furthermore, our evaluations show that in videos and large files with a small amount of changes, redundant chunks are likely to reside in very long chains that are efficiently handled by a receiver-based TRE.

Traffic volume and detected redundancy

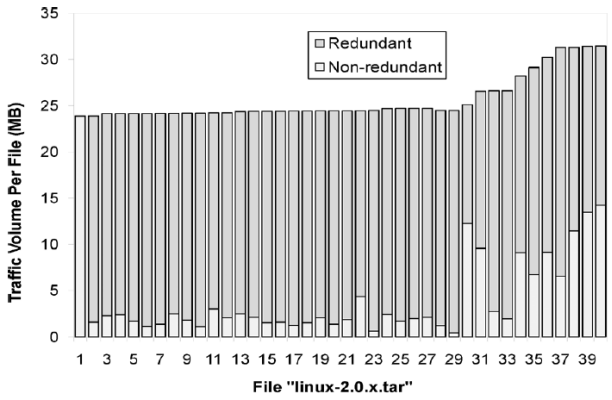

Fig. 4. (a) Linux source: 40 different Linux kernel versions.

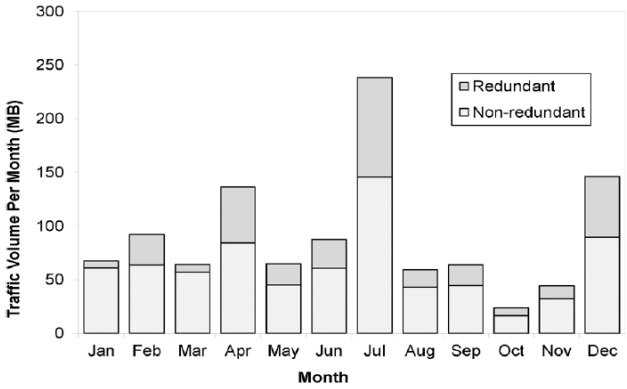

Fig. 4. (b) Email: 1-year Gmail account by month.

Table I Cloud Operational Cost Comparison

	No TRE	PACK	Server-based
Traffic volume	9.1 TB	6.4 TB	6.2 TB
Traffic cost reduction (Figure 4)		30%	32%
Server-hours cost increase (Figure 9)		6.1%	19.0%
Total operational cost	100%	80.6%	83.0%

Table I summarize the costs and the benefits of the TRE operations and compare them to a baseline with no TRE. Both TRE schemes identify and eliminate the traffic redundancy. However, while PACK employs the server only when redundancy exists, the sender-based TRE employs it for the entire period of time, consuming more servers than PACK and no-TRE schemes when no or little redundancy is detected.

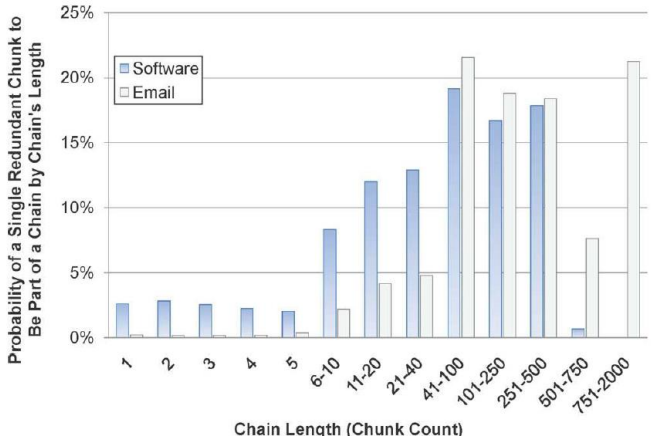

Fig. 5 Chain length histogram Linux Software and Email data collections

IMPLEMENTATION

Our implementation enables the transparent use of the TRE at both the server and the client. PACK receiver–sender protocol is embedded in the TCP Options field for low overhead and compatibility with legacy systems along the path. We keep the genuine operating systems' TCP stacks intact,

allowing a seamless integration with all applications and protocols above TCP. Chunking and indexing are performed only at the client's side, enabling the clients to decide independently on their preferred chunk size.

In our implementation, the client uses an average chunk size of 8 KB. We found this size to achieve high TRE hit-ratio in the evaluated datasets, while adding only negligible overheads of 0.1% in metadata storage and 0.15% in predictions bandwidth.

Server Operational Cost

We measured the server performance and cost as a function of the data redundancy level in order to capture the effect of the TRE mechanisms in real environment. To isolate the TRE operational cost, we measured the server's traffic volume and CPU utilization at maximal throughput without operating a TRE.

As the redundancy grows, the PACK server cost decreases due to the bandwidth saved by unsent data. However, the EndRE-like server does not gain a significant cost Decrement since the SHA-1 operations are performed over nonredundant data as well. Note that at above 25% redundancy, which is common to all reviewed datasets, the PACK operational cost is at least 20% lower than that of EndRE-like.

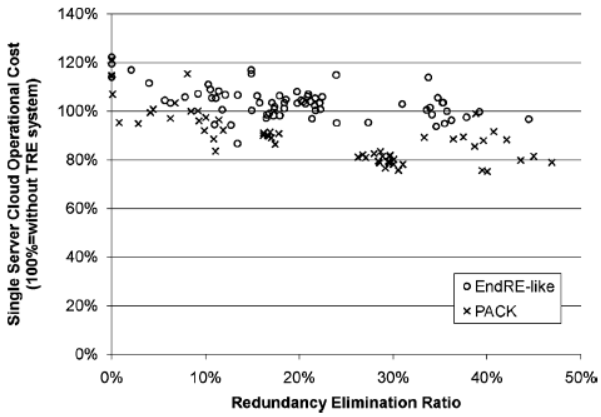

Fig. 6. PACK versus EndRE-like cloud server operational cost as a function of redundancy ratio.

PACK Impact on the Client CPU

To evaluate the CPU effort imposed by PACK on a client, we measured a random client under a scenario similar to the one used for measuring the server's cost, only this time the cloud server streamed videos at a rate of 9 Mb/s to each client. Such a speed throttling is very common in real-time video servers that aim to provide all clients with stable bandwidth for smooth view.

The average PACK-related CPU consumption of a client is less than 4% for 9-Mb/s video with 36.4% redundancy.

Chunking Scheme

Pack an XOR-based rolling hash function, tailored for fast TRE chunking. Anchors are detected by the mask in line that provides on average 8-kB chunks.

PACK chunking algorithm

1. Mask 0x00008A3110583080 {48 bytes window; 8 KB chunks}
2. {has to be 64 bits}
3. for all byte stream do
4. shift left *longval* by 1 bit { ; drop msb}
5. bitwise-xor *byte*
6. if processed at least 48 bytes and (*longval* bitwise-and *mask*) then
7. found an anchor
8. end if
9. end for

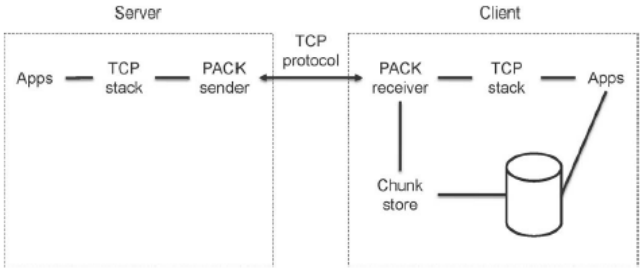

Fig7. Overview of PACK Implementation

We have to present the Predictive ACK hierarchal mode of operation. PACK computes the data dispersion value using an exponential smoothing function

$$D \leftarrow D\alpha+(1-\alpha)M$$

Where α is a smoothing factor. The value is set to 0 when a chain break is detected and 255 otherwise.

In our implementation, the client uses an average chunk size of 8 KB. We found this size to achieve high TRE hit-ratio in the evaluated datasets, while adding only negligible overheads of 0.1% in metadata storage and 0.15% in predictions bandwidth. For the experiments held in this section, we generated datasets: IMAP e-mails, HTTP videos, and files downloaded over FTP. The workload was then loaded to the server and consumed by the clients. We sampled the machines' status every second to measure real and virtual traffic volumes and CPU utilization.

Interconnect communication

The interconnect communication is critical for the design of AppDedupe. We detail the operations carried out when storing and retrieving a file. A file store request a client sends a *PutFileReq* message to the director after file partitioning and chunk fingerprinting. The message includes file metadata like: file ID (the SHA-1 value of file content), file size, file name, timestamp, the number of super-chunk in the file and their checksums.

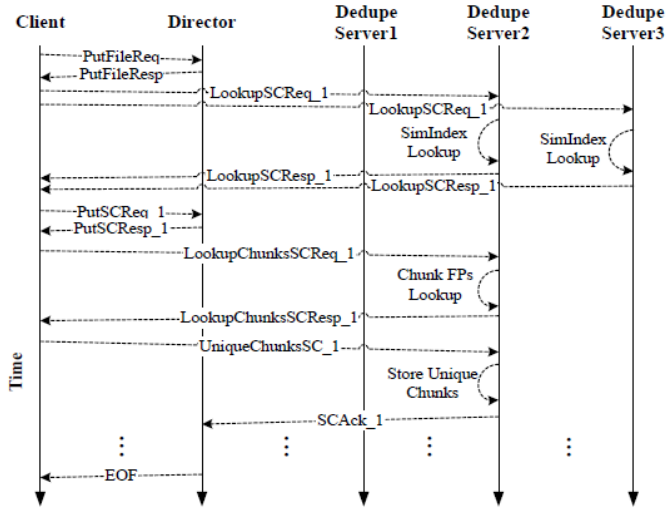

Fig.8. Message exchanges for store operation

The director reacts to this request by querying the file recipe, and forwards the GetFileResp message to the client. The GetFileResp contains the super-chunk list in the file and the mapping from super-chunk to the dedupe stor-age node where it is stored. Then, the client requests each super-chunk in the file from the corresponding dedupe storage node with GetSuperChunk message. The dedupe server can retrieve super-chunk from data containers, and the performance of restore process can be accelerated. Finally, the client downloads each super-chunk and uses the checksums of super-chunks and file ID to verify the data integrity.

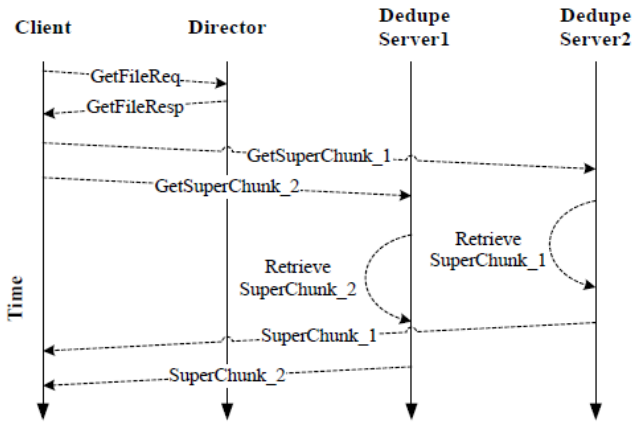

Fig.9. Message exchanges for retrieve operation

CONCLUSION

Cloud computing is expected to trigger high demand for TRE solutions as the amount of data exchanged between the cloud and its users is expected to dramatically increase. The cloud environment redefines the TRE system requirements, making proprietary middle-box solutions inadequate. Consequently, there is a rising need for a TRE solution that reduces the cloud's operational cost while accounting for application latencies, user mobility, and cloud elasticity.

In this paper, we have presented PACK, a receiver-based, cloud-friendly, end-to-end TRE that is based on novel speculative principles that reduce latency and cloud operational cost. PACK does not require the server to continuously maintain clients' status, thus enabling cloud elasticity and user mobility while preserving long-term redundancy. Moreover, PACK is capable of eliminating redundancy based on content arriving to the client from multiple servers without applying a three-way handshake.

REFERENCES

[1]. E. Zohar, I. Cidon, and O. Mokryn, "The power of prediction: Cloud bandwidth and cost Decrement," in Proc. SIGCOMM, 2011, 86–97.

[2]. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph,R.Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Commun. ACM, 53(4), 50–58, 2010.

[3]. U. Manber, "Finding similar files in a large file system," in Proc.USENIX Winter Tech. Conf., 1994, 1–10.

[4]. N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in Proc. SIGCOMM, 30, 2000, 87–95.

[5]. A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in Proc. SOSP, 2001, 174–187.
[6]. E. Lev-Ran, I. Cidon, and I. Z. Ben-Shaul, "Method and apparatus for reducing network traffic over low bandwidth links," US Patent 7636767, 2009.
[7]. S.Mccanne andM. Demmer, "Content-based segmentation scheme for data compression in storage and transmission including hierarchical segment representation," US Patent 6828925, 2004.
[8]. R. Williams, "Method for partitioning a block of data into subblocks and for storing and communicating such subblocks," US Patent 5990810, 1999.
[9]. Juniper Networks, Sunnyvale, CA, USA, "Application acceleration," [Online]. Available. 1996. http://www.juniper.net/us/en/products-services/application-acceleration/