International Journal of Computer Sciences
February'2017 Issue
ISSN Number: 4142 3453

PREDICTIVE ACK'S FOR END TO END TRAFFIC REDUNDANCY ELIMINATION SYSTEM

Bandela Rajesh[1] ,V.Manohar[2]

[1]M.Tech (CSE)., Dept of CSE., VAAGDEVI INSTITUTE OF TECHNOLOGY &SCIENCE,Peddasettipalli(v),Proddatur
[2]Assistant Professor, Dept of CSE, VAAGDEVI INSTITUTE OF TECHNOLOGY &SCIENCE,Peddasettipalli(v),Proddatur

Abstract

In this paper, we present PACK (Predictive ACKs), a novel end-to-end traffic redundancy elimination (TRE) system, designed for cloud computing customers. Cloud-based TRE needs to apply a judicious use of cloud resources so that the bandwidth cost reduction combined with the additional cost of TRE computation and storage would be optimized. PACK's main advantage is its capability of offloading the cloud- server TRE effort to end-clients, thus minimizing the processing costs induced by the TRE algorithm. Unlike previous solutions, PACK does not require the server to continuously maintain clients' status. This makes PACK very suitable for pervasive computation environments that combine client mobility and server migration to maintain cloud elasticity. PACK is based on a novel TRE technique, which allows the client to use newly received chunks to identify previously received chunk chains, which in turn can be used as reliable predictors to future transmitted chunks. We present a fully functional PACK implementation, transparent to all TCP-based applications and net-work devices. Finally, we analyze PACK benefits for cloud users, using traffic traces from various sources.

Keywords

Caching, Cloud Computing, Network Optimization, Traffic Redundancy Elimination

I. Introduction

Cloud computing offers its customers an economical and convenient pay-as-you-go service model, known also as usage-based pricing [2]. Cloud customers1 pay only for the actual use of computing resources, storage, and band-width, according to their changing needs, utilizing the cloud's scalable and elastic computational capabilities. In particular, data transfer costs (i.e., bandwidth) is an important issue when trying to minimize costs [2]. Consequently, cloud customers, applying a judicious use of the cloud's resources, are motivated to use various traffic reduction techniques, in particular traffic redundancy elimination (TRE), for reducing bandwidth costs.

Traffic redundancy stems from common end-users' activities, such as repeatedly accessing, downloading, uploading (i.e., backup), distributing, and modifying the same or similar information items (documents, data, Web, and video). TRE is used to eliminate the transmission of redundant content and, there-fore, 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 [3–5]. Commercial TRE solu-tions 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 (e.g., Cisco [6], Riverbed [7], Quantum [8], Juniper [9], Blue Coat [10], Expand Networks [11], and F5 [12]).

In this paper, we present a novel receiver-based end-to-end TRE solution that relies on the power of predictions to eliminate redundant traffic between the cloud and its end-users. In this solution, each receiver observes the incoming stream and tries to match its chunks with a previously received chunk chain or a chunk chain of a local file. Using the long- term chunks' meta-data information kept locally, the receiver sends to the server predictions that include chunks' signatures and easy-to-verify hints of the sender's future data. The sender first examines the hint and performs the TRE operation only on a hint- match. The purpose of this procedure is to avoid the expensive TRE com-putation at the sender side in the absence of traffic redundancy. When redundancy is detected, the sender then sends to the receiver only the ACKs to the predictions, instead of sending the data.

II. Related Work

Several TRE techniques have been explored in recent years. A protocol -independent TRE was proposed in [4]. The paper describes a packet-level TRE, utilizing the algorithms presented in [3].

Several commercial TRE solutions described in [6] and [7] have combined the sender-based TRE ideas of [4] with the algorithmic and implementation approach of [5] along with protocol specific optimizations for middle-boxes solutions. In particular, [6] describes how to get away with three-way handshake between the sender and the receiver if a full state synchronization is maintained.

III. Pack Algorithm

For the sake of clarity, we first describe the basic receiver-driven operation of the PACK protocol. Several enhancements and optimizations are introduced in Section IV.

A. Receiver Chunk Store

PACK uses a new chains scheme, described in Fig. 1, in 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.

B. 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).

International Journal of Computer Sciences
February'2017 Issue
ISSN Number: 4142 3453

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

Proc. 2: predAttempt()
8 if received chunk matches one in chunk store then
9 if foundChain(chunk) then
10 prepare PREDs
11 send single TCP ACK with PREDs according to Options free space
12 exit
13 end if
14 else
15 store chunk
16 link chunk to current chain
17 end if
18 send TCP ACK only

Proc. 3: processPredAck()
for all offset ∈ PRED-ACK do
read data from chunk store
put data in TCP input buffer
end for

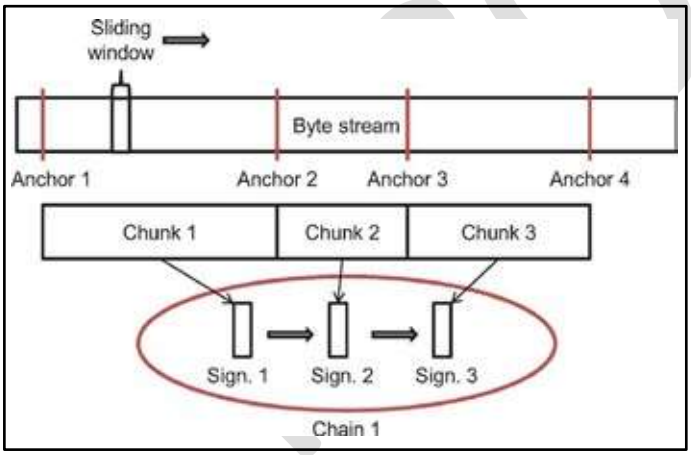

Fig. 1: From Stream to Chain

C. 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.

D. Wire Protocol

In order to conform with existing firewalls and minimize overheads, we use the TCP Options field to carry the PACK wire protocol. It is clear that PACK can also be implemented above the TCP level while using similar message types and control fields.

IV. Optimizations

For the sake of clarity, Section III presents the most basic version of the PACK protocol. In this section, we describe additional options and optimizations.

A. Adaptive Receiver Virtual Window

PACK enables the receiver to locally obtain the sender's data when a local copy is available, thus eliminating the need to send this data through the network. We term the receiver's fetching of such local data as the reception of virtual data.

Proc. 4: predAttemptAdaptive()—obsoletes Proc. 2

1. {new code for Adaptive}
2. if received chunk overlaps recently sent prediction then
3. if received chunk matches the prediction then
4. predSizeExponent()
5. else
6. predSizeReset()
7. end if
8. end if
9. if received chunk matches one in signature cache then
10. if foundChain(chunk) then
11. {new code for Adaptive}
12. prepare PREDs according to predSize
13. send TCP ACKs with all PREDs
14. exit
15. end if
16. else
17. store chunk
18. append chunk to current chain
19. end if
20. send TCP ACK only

B. Cloud Server as a Receiver

In a growing trend, cloud storage is becoming a dominant player [13-14]—from backup and sharing services [5] to the American National Library [6], and e -mail services [7-8]. In many of these services, the cloud is often the receiver of the data.

International Journal of Computer Sciences
February'2017 Issue
ISSN Number: 4142 3453

C. 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. To that end, we present the PACK hybrid mode of operation, described in Proc. 6 and Proc. 7. When PACK recognizes a pattern of dispersed changes, it may select to trigger a sender-driven approach in the spirit of [4], [6-7], and [12].

V. 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.

VI. Implementation

In this section, we present PACK implementation, its performance analysis, and the projected server costs derived from the implementation experiments.

Our implementation contains over 25 000 lines of C and Java code. It runs on Linux with Net filter Queue [3].The PACK implementation architecture. At the server side, we use an Intel Core 2 Duo 3 GHz, 2 GB of RAM, and a WD1600AAJS SATA drive desktop. The clients laptop machines are based on an Intel Core 2 Duo 2.8 GHz, 3.5 GB of RAM, and a WD2500BJKT SATA drive.

A. 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. We then used these numbers as a reference cost, based on present Amazon EC2 [9] pricing. The server operational cost is com-posed of both the network traffic volume and the CPU utiliza-tion, as derived from the EC2 pricing.

B. 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.

C. Pack Messages Format

In our implementation, we use two currently unused TCP option codes, similar to the ones de fined in SACK [2]. The first one is an enabling option PACK permitted sent in a SYN segment to indicate that the PACK option can be used after the connection is established. The other one is a PACK message that may be sent over an established connection once permission has been granted by both parties.

VII. 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 en-vironment redefines the TRE system requirements, making pro-prietary middle -box solutions inadequate. Consequently, there is a rising need for a TRE solution that reduces the cloud's op-erational 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 specula-tive 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.

Our evaluation using a wide collection of content types shows that PACK meets the expected design goals and has clear advan-tages over sender -based TRE, especially when the cloud com-putation

cost and buffering requirements are important. More-over, PACK imposes additional effort on the sender only when redundancy is exploited, thus reducing the cloud overall cost.

Two interesting future extensions can provide additional ben-efits to the PACK concept. First, our implementation maintains chains by keeping for any chunk only the last observed sub-sequent chunk in an LRU fashion. An interesting extension to this work is the statistical study of chains of chunks that would enable multiple possibilities in both the chunk order and the corresponding predictions. The system may also allow making more than one prediction at a time, and it is enough that one of them will be correct for successful traffic elimination. A second promising direction is the mode of operation optimization of the hybrid sender–receiver approach based on shared decisions de-rived from receiver's power or server's cost changes.

References

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

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia,"A view of cloud computing", Commun. ACM, Vol. 53, No. 4, pp. 50–58, 2010.

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

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

[5] A. Muthitacharoen, B. Chen, D. Mazières,"A low-bandwidth net-work file system", In Proc. SOSP, 2001, pp. 174–187.

[6] E. Lev-Ran, I. Cidon, I. Z. Ben-Shaul,"Method and apparatus for reducing network traffic over low bandwidth links", US Patent 7636767, Nov. 2009.

[7] S. Mccanne and M. Demmer, "Content-based segmentation

scheme for data compression in storage and transmission including hierarchical segment representation", US Patent 6828925, Dec. 2004.

[8] R. Williams,"Method for partitioning a block of data into subblocks and for storing and communicating such subblocks", US Patent 5990810, Nov. 1999.

[9] Juniper Networks, Sunnyvale, CA, USA,"Application acceleration", 1996 [Online] Available: http://www.juniper.net/us/en/products-services/application-acceleration/

[10] Blue Coat Systems, Sunnyvale, CA, USA,"MACH5", 1996 [Online] Available: http://www.bluecoat.com/products/mach5