



## Cloud Computing – Final Project Submission report

### **Team Members**

Sridevi Mallipudi (16251191)

Pragathi Thammaneni (16230695)

## DATA DEDUPLICATION BY END -TO-END TRAFFIC REDUNDANCY ELIMINATION

### Introduction:

Data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data often called intelligent compression or single-instance storage. This technique is used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent. In the deduplication process, unique chunks of data, or byte patterns, are identified and stored during a process of analysis. Deduplication techniques take advantage of data similarity to identify the same data and reduce the storage space.

In that way, data deduplication closely aligns with incremental backup, which copies only the data that has changed since the previous backup.

### Motivation:

The pay-as-you-go service model impels cloud customers to reduce the usage cost of bandwidth. Traffic Redundancy Elimination (TRE) has been shown to be an effective solution for reducing bandwidth costs, and thus has recently captured significant attention in the cloud environment.

By studying the TRE techniques in a trace driven approach, we found that both short-term (time span of seconds) and long-term (time span of hours or days) data redundancy can concurrently appear in the traffic, and solely using either sender-based TRE or receiver-based TRE cannot simultaneously capture both types of traffic redundancy. Also, the efficiency of existing receiver-based TRE solution is susceptible to the data changes compared to the historical data in the cache.

A Cooperative end-to-end TRE solution (CoRE) has been proposed which can detect and remove both short-term and long-term redundancy through a two-layer TRE design with cooperative operations between layers. An adaptive prediction algorithm is further proposed to improve TRE efficiency through dynamically adjusting the prediction window size based on the hit ratio of historical predictions.

**Advantage:** CoRE - to adapt to different traffic redundancy characteristics of cloud applications to improve its operation cost.

## Proposed System:

This paper proposes a Cooperative end-to-end TRE CoRE, which integrates two-layer TRE efforts and several optimizations for TRE efficiency.

Reference: IEEE paper by Lei Yu, Haiying Shen, Karan Sapra, Lin Ye, and Zhipeng Cai

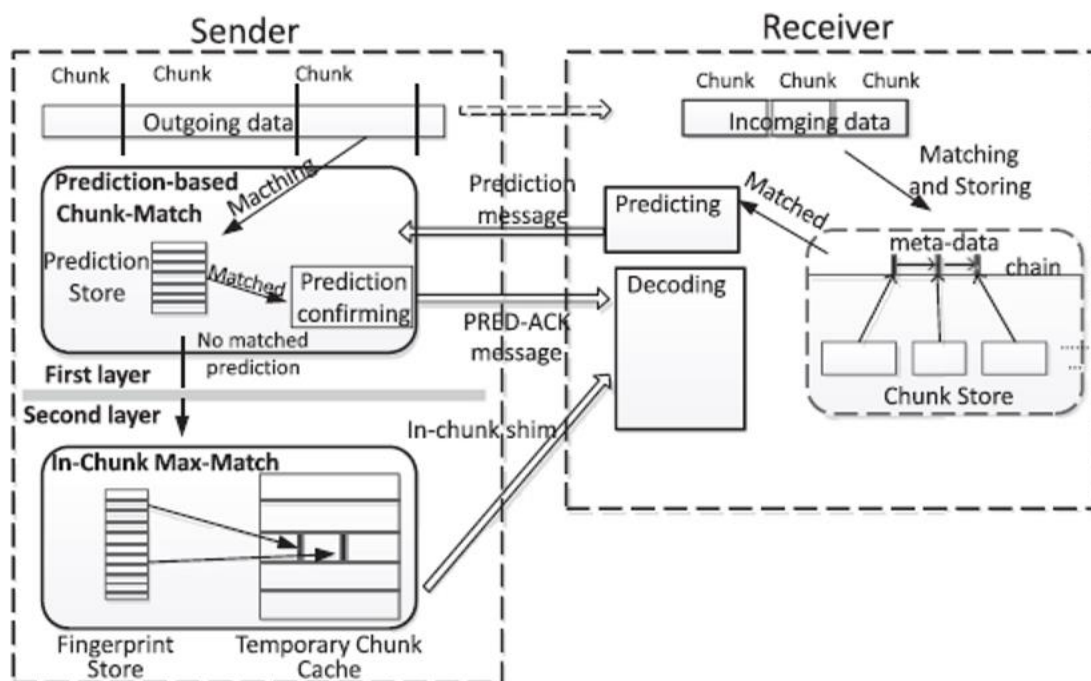
<http://ieeexplore.ieee.org/document/7488173/>

CoRE has two TRE modules each for capturing short-term redundancy and long-term redundancy respectively. Two TRE modules are integrated into a two-layer redundancy detection system. For any outbound traffic from the server, CoRE first detects the long-term redundancy by the firstlayer TRE module. If no redundancy is found, it turns to the second-layer TRE module to search for short-term redundancy at finer granularity.

The first-layer TRE module detects long-term redundancy by a prediction-based Chunk-Match approach like PACK

In the second-layer TRE module, both the server and client maintain a temporary small local chunk cache to store most recently transmitted and received chunks during their communication respectively.

## Architecture:



Reference: <http://ieeexplore.ieee.org/document/7488173/>

## Software's Used:

IDE: Net Beans 8.2

Programming Language: Java

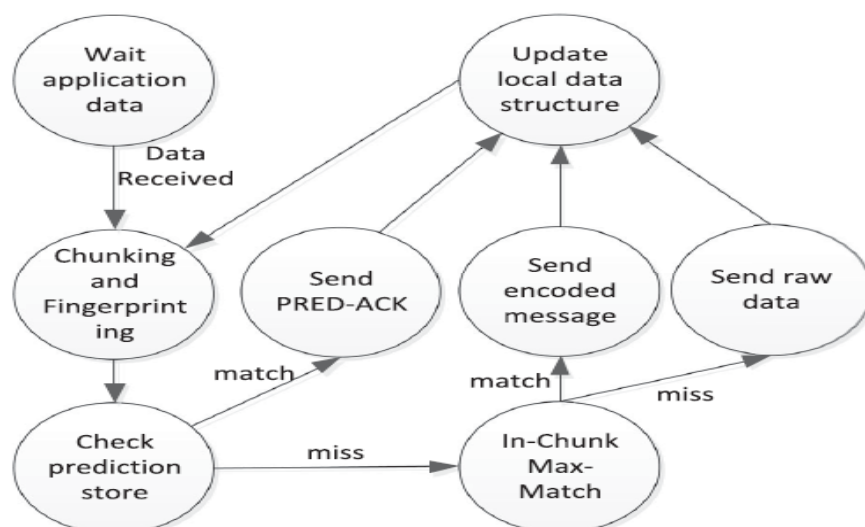
## Detailed Implementation:

### 1. Chunking and Fingerprinting

CoRE coordinately uses prediction based Chunk-Match and In-Chunk Max-Match to implement two-layer TRE in order to maximally detect redundancy by capturing both long-term and short-term redundancy. Chunk-Match identifies the repeated chunks in a TCP stream while Max-Match identifies the maximal repeated substrings in a chunk.

### 2. Sender

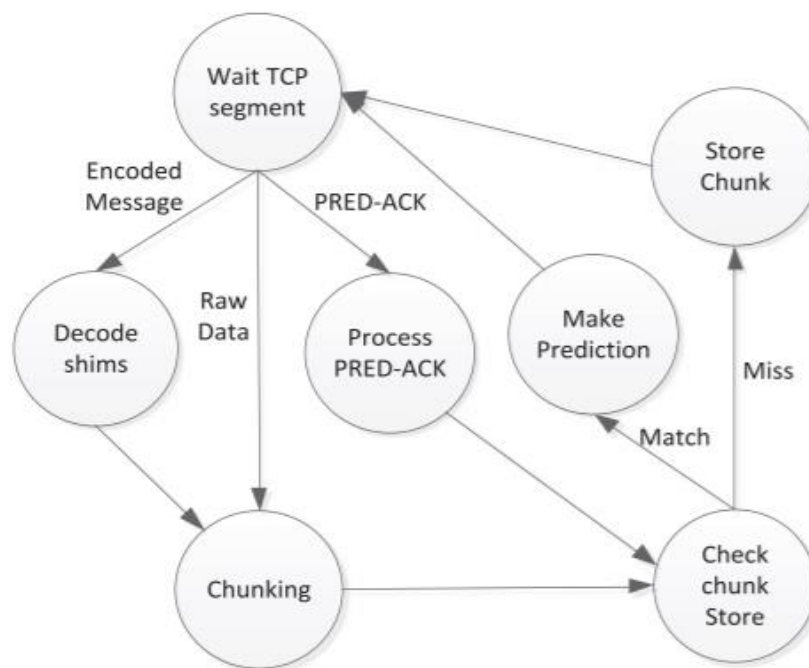
The sender uses a prediction store to cache the most recent predictions received from the receiver. Each prediction contains the SHA-1 signature of a predicted chunk and its expected offset, i.e., TCP sequence number in the TCP byte stream. The sender also has a chunk cache to store chunks that have been recently sent. A fingerprint store holds the meta-data of every representative fingerprint for each cached chunk, which includes the fingerprint value, the address of the chunk in the cache referred by the fingerprint, and the byte offset of the window in the chunk over which the fingerprint is computed.



Reference: <http://ieeexplore.ieee.org/document/7488173/>

### 3. Receiver

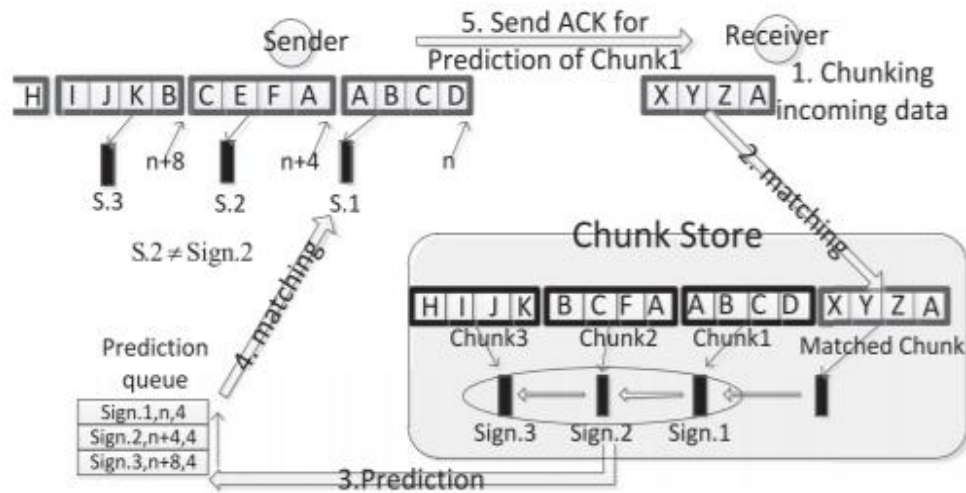
For each TCP connection, the receiver divides the incoming data stream into chunks and maintains a local prediction store which caches recent predictions for the TCP connection. To reconstruct the raw data stream, the receiver processes incoming TCP segments according to their different types. There are three types of TCP segments from the sender to the receiver: PRED-ACK message, the message encoded with shims and raw data. The chunk predictions are sent within a PRED message and in the order of their expected offsets.



Reference: <http://ieeexplore.ieee.org/document/7488173/>

#### Development and Deployment:

A web application has been developed that captures the data from the sender and forwards it to the receiver. At the receiver, the incoming TCP data stream is divided into chunks. The chunks are linked in sequence, which forms a chain, and are stored into a local chunk store. The receiver compares each incoming chunk to the chunk store. Once finding a matching chunk on a chain, it retrieves a number of subsequent chunks along the chain as the predicted chunks in the future incoming data.



Reference: <http://ieeexplore.ieee.org/document/7488173/>

The signatures of the retrieved chunks and their expected offsets in the incoming data stream are sent in a PRED message to the sender as a prediction for the sender's subsequent outgoing data. To match data with a prediction, the sender computes SHA-1 over the outgoing data at the expected offset with the length given by the prediction, and compares the result with the signature in the prediction. Upon a signature match, the sender sends a PRED-ACK message to the receiver to tell it to copy the matched data from its local storage.

This implementation avoids the additional computational and storage costs incurred by TRE at the cloud to outweigh the bandwidth saving gains.

### Screenshots:


The core implementation of this project includes Admin, Core Sender – User, Core Receiver – Server. Below are the screenshots which gives the outline of the application:

**Admin:**

Cloud Server Details

Cloud Server Id

1



**Server:**

Connect Connected User Id Chunk Store Uploaded Data

Connected User Id

1

**User:**

Connect with Server Generate Chunks Generate Signatures Prediction Queue Second Layer


Data: A B C D E F G H I J K L

View Selected Data: ABCDEFGHIJKL

Split Data into Chunks

Clear

ABCD  
EFGH  
IJKL



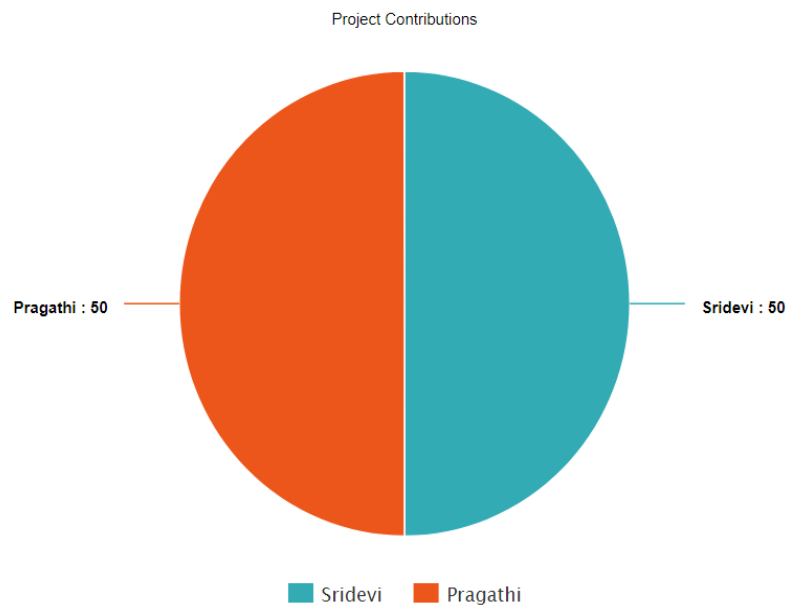
Please refer the below github link for project screenshots.

<https://github.com/PragathiThammaneni/Cloud-Computing/blob/master/Final%20Project/Dedup-Cloud.pptx>

#### Challenges Faced:

- Faced issues while generating the hash values.
- Calculating the fingerprints for sub window lengths in second layer.

#### Contributions:



#### Code:

Please refer the below github link for code

<https://github.com/PragathiThammaneni/Cloud-Computing/tree/master/Final%20Project/Code>