Deep learning based content caching

Software Requirements Specification

1.0

25th January 2019

Team-J

Ajaydeep - 170001003
Pavan - 170001016
Jhalak Gupta - 170001024
Mrigank Krishan - 170001030

Prepared for
CS 258 Software Engineering
Spring 2019

Revision History

Date	Description	Author	Comments
25-01-19	Version 1		First Revision

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

1. Introduction

1.1 Purpose

The aim of this document is to specify the complete description of the project on Deep learning based content caching framework. This document describes the functionalities, external interfaces, attributes and the design constraints of the system which will be developed as part of this project. It is intended to be used by members of the project team that will implement and verify the correct functioning of the system. The client can also use this document to verify the fulfillment of his/her/their requirement.

1.2 Scope

1) Content popularity prediction.
2) Personalized recommendation engine.

1.3 Definitions, Acronyms, and Abbreviations

Python: Python is an interpreted, high-level, general-purpose programming language

DL: Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

LSTM: Long short-term memory units are units of a recurrent neural network. An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.

NN: Artificial neural networks or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains

RNN: A recurrent neural network is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state to process sequences of inputs

Keras: Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano or PlaidML . Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Tensorflow: TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

CRNN: Convolutional recurrent neural network

CNN: Convolutional neural network

CDN: Content Delivery Network

QOE: Quality of Experience

MITM: Man in the middle

SSL: Secure socket layer

GPU: Graphics Processing Unit

TPU: Tensor Processing Unit

1.4 References

1) "DeepCache: A Deep Learning Based Framework For Content Caching" published on 2018-08-07. Source: https://dl.acm.org/citation.cfm?doid=3229543.3229555

2) "DeepMEC: Mobile Edge Caching Using Deep Learning" Received October 9, 2018, accepted November 13, 2018, date of current version December 31, 2018. Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8576500

1.5 Overview

The rest of the SRS document is divided into four main parts:

1) **General Requirements:** This section describes the general requirements for efficient and convenient usage of application. It is an attempt to make understanding the requirements easier.

2) **Specific Requirements:** This section guides through the requirements specific to the design and implementation of this application. A brief overview of the hardware, software and interface requirements is provided in this section.

3) **Functional Requirements:** This includes functional requirements which are to be fulfilled by the application being developed. Basic features of the application are described here.

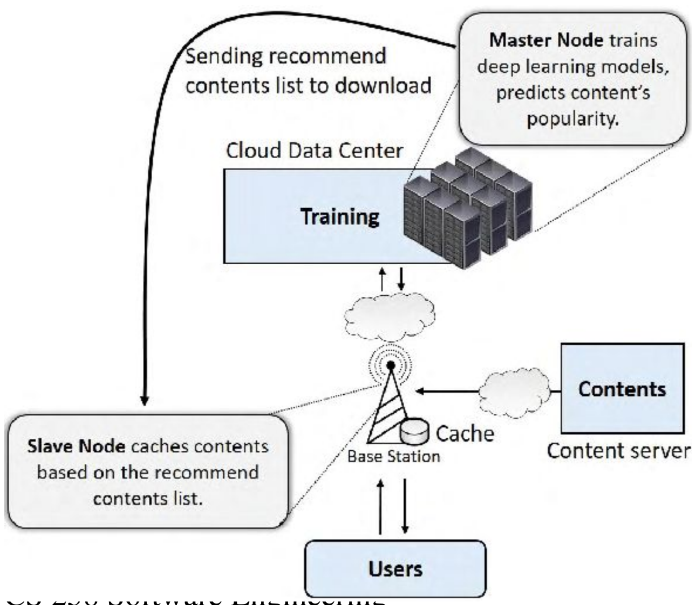
4) Non-Functional Requirements: Non functional requirements are the performance characteristics of the system. These include requirements like speed, security, recoverability etc.

2. General Description

2.1 Product Utility

Content objects are heterogeneous as they vary in size (e.g., webpages vs. videos), access pattern, and popularity. A study shows that 70% of objects served by a cache server are requested only once over a period of days. Object access patterns are frequently changing due to the frequent changes in object popularity. Object popularity changes within each day according to the diurnal pattern, and also over days according to the object's life span. In addition, changes in request routing algorithms due to network/server failures can also cause changes in object access patterns. Due to these frequent changes, the assumption of stationary object access patterns becomes invalid. Thus, caching algorithms cannot rely on the locally observed object access patterns for making decisions. On the other hand, manually tuning the caching algorithm for each cache server according to the changes of request access patterns is very expensive and is not scalable.
Thus, if we know ahead of time an estimation for object characteristics, we can utilize such information in the caching mechanism to cope with the predicted changes. The cache performance depends on the prediction accuracy, and how it is being utilized to make decisions.

2.2 Product Perspective


Sending recommend contents list to download
Master Node trains deep learning models, predicts content's popularity.
Cloud Data Center
Training
Contents
Content server
Cache
Base Station
Slave Node caches contents based on the recommend contents list.
Users

The proposed system model is shown in Fig. in which the central controller (Master Node) is implemented at the high-end computing node (e.g., cloud data center) and the Slave Node is implemented at the BS. The Master Node is responsible for training the deep learning model by utilizing collected data from the BS, then predicts the future popularity of contents with trained models and sends the contents list to the BS to cache popular contents. The Slave Node is responsible for storing contents recommended by the Master Node and data collection (which collects information such as request counts and the number of cache hits at the BS). If the requested content is located at the BS's cache, the BS immediately replies it to the users. Otherwise, the requested content is retrieved from the content server and then provided to the users.

2.3 Product Goal

Our goal is to develop a self-adaptive caching mechanism, which automatically learns the changes in request traffic patterns, especially bursty and non-stationary traffic, and predicts future content popularity, then decides which objects to cache and evict accordingly to maximize the cache hit. In recent years, recurrent neural networks (RNN) have become the cornerstone for sequence prediction. RNNs have shown their unchallenged dominance in the area of natural language processing, machine language translation, speech recognition, and image captioning. Many variants of RNN exist in literature, among which Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) are the most popular ones for sequence prediction. Thus, it is natural to wonder their ability to predict content popularity where content requests arrive in a form of a sequence.

2.4 Product Functions

1. Predicting the future popularity score of contents based on previously viewed data.
2. Caching the predicted contents with high popularity scores.
3. A mobile app to show popular and trending content in a local network.

2.5 Challenges

This task has many challenges:

(1) The future object characteristics need to be forecasted to be available at the time of making cache decisions.

(2) These characteristics change over time, and hence, this forecasting needs to run continuously.

(3) The caching mechanism needs to carefully utilize these predicted object characteristics to improve cache performance.

(4) It requires high computation resources to process the big data (high-dimensional data) to train the prediction models.

(5) Because, our solution is basically a MITM and SSL is encrypted, we decided not to cache SSL content to improve user experience.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Capability to cache popular content

3.1.1.1 Introduction

It offers new primitives such as in-network caching, in which storage becomes an integral part of the network substrate (i.e., routers have the capability to cache objects on-the-$y, and serve user requests for cached objects). This makes caching algorithms a major aspect in video streaming applications. Without caching, every user request is fetched from the backend/origin server, which increases the network load, as well as user-perceived latency. As a result, user engagement is impacted, which leads to signi!cant revenue loss for content providers

3.1.2.2 Inputs

We provide multiple past probabilities, the number of requests for a particular content as input.

3.1.2.3 Processing

In our evaluation, for every object request, we predict popularity of all objects for different time intervals in future. This information is then used by the Caching Policy component to make decisions that control the caching behavior. For example, a Caching Policy could control what objects to cache and evict.
Content popularity can be defined as the ratio of the number of requests for a particular content to the total number of requests, usually obtained for a certain region during a given period of time.
Each of these probabilities are calculated using a predefined probability window. As a result, our input and output can be seen as a 3D Tensor with dimension. We construct the input and output tensor from a given trace of object requests (i.e., a workload) and split the data further into training and testing parts. Finally, we train our model to predict future probabilities of any requested object at time t.

3.1.2.4 Outputs

1) Reduced load on cloud data server
2) Improved network performance

3.1.1.2 Error Handling

Object popularity changes within each day according to the diurnal pattern, and also over days according to the object's life span. In addition, changes in request routing algorithms due to network/server failures can also cause changes in object access patterns. Due to these frequent changes, the assumption of stationary object access patterns becomes invalid. Thus, caching algorithms cannot rely on the locally observed object access patterns for making decisions. On the other hand, manually tuning the caching algorithm for each cache server according to the changes of request access patterns is very expensive and is not scalable.

3.1.2 Improved Quality of Experience (QoE)

3.1.2.1 Introduction

By 2021, 77% of the Internet video traffic is expected to cross CDNs. Hence, adding cache storage space at routers becomes of utmost importance to handle this massive growth, and improve the network performance as well as user's QoE. Thus, utilizing caching capabilities at the edge network reduces the video content's access delay or the number of videos retrievals from the original content server by storing popular videos, where the cache capacity of the edge network is limited.

3.1.2.4 Outputs

1) Access delay minimization which provides the contents to the user with no delay.
2) Rapid increase in video streaming services due to reduced buffering.

3.1.2.5 Error Handling

The main challenge to designing an efficient caching policy is predicting the future popularity of video contents beforehand, since the video's popularity varies both temporally and spatially

3.2 Non-Functional Requirements

3.2.1 Performance

Since deep learning require high processing power. Hence algorithm must be efficient enough to process large no of data set.

3.2.2 Reliability

The system should be able to process all the information for analytics and store them permanently as any loss of data will decrease the accuracy of the system..

3.2.3 Availability

The system should have a downtime not greater than a few hours during which the caching may not be performed.

3.2.4 Security

The data in the slave node can only be written by the master node. Hence, an unauthorized person can not spoof the cache.

3.2.5 Maintainability

The database should be accessible to the administrators so as to carry out maintenance. The database should also be periodically backed up to prevent information loss due to system crashes

3.3 Inverse Requirements

- It should not slow down the network.

3.4 Logical Database Requirements

- A database is required to store information such as cache hits. It will also log each and every request made to the slave node which will be used by our algorithm to make make further predictions.
- At least 10 GB storage is recommended for database storage requirements.

3.5 Other Requirements

- A server with good processing power
- GPU/TPU for faster processing

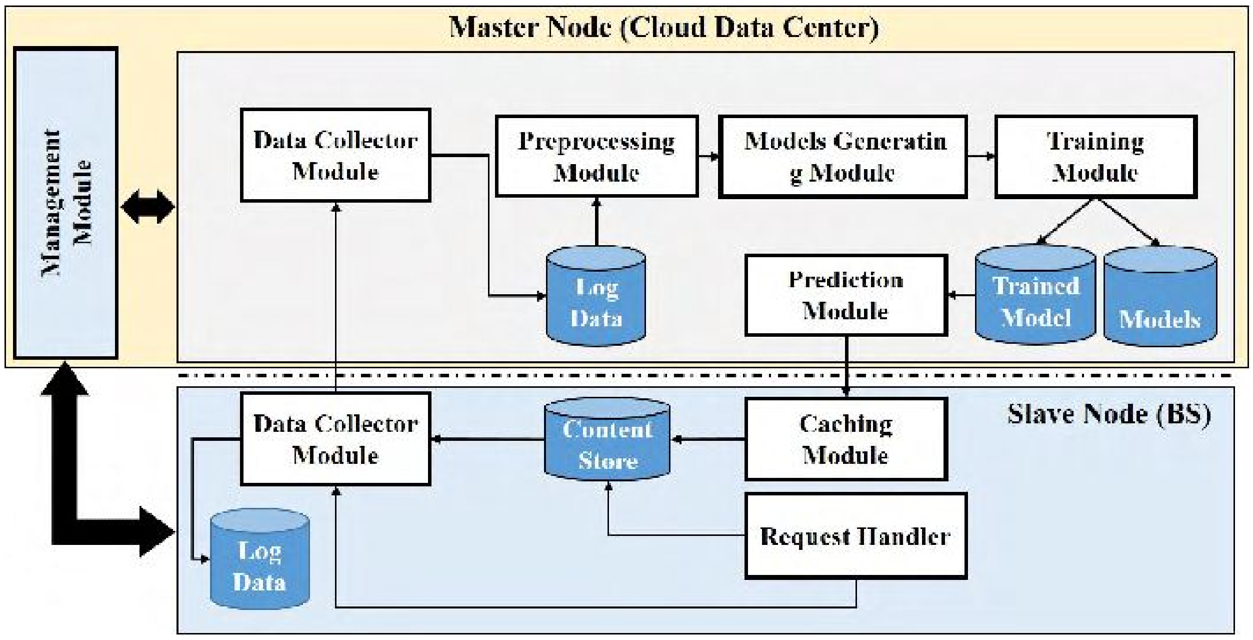


Fig. An overview system design for learning-based caching