The Chinese University of Hong Kong, Shenzhen

ECE4016

Computer Network

Assignment 2 Report

Author:
Yang Hongmeng

Student Number:
119010378

November 12, 2022

Contents

1 Introduction

In this assignment, I implement ABR algorithm by following the paper "Probe and adapt: Rate adaptation for HTTP video streaming at scale. (2014)". Before introducing my implementation detail, I will introduce the background here.

Adaptive bitrate streaming (ABR) is a method for improving streaming over HTTP networks. The term "bitrate" refers to how quickly data travels across a network and is often used to describe an Internet connection's speed. In this algorithm, the video steam would be segmented into several chunks. The server would also packet and send each chunk with different bitrate one time.

However, the Internet environment changes all the time. Therefore, in order to make sure the quality of video and the experience of client, the client side server would choose which bitrate version of video chunk would be download according to current Internet condition.

Most of algorithm focus on how to make both video quality and the download speed good for client to use. The final target is to make the video quality as high as possible and the flow of video as fluent as possible.

In this report, I will show how conventional approach solve this problem and how I optimize this algorithm.

The detailed implementation methods and explanation are shown below.

2 Algorithm implementation

2.1 Theoretical implementation

The conventional approach has three steps:

First step is estimating the current bandwidth according to last time throughput. According to the given data, we can directly get the previous time throughput.

The second step is smoothing the estimated bandwidth by utilizing the smooth function.

The last step is returning the bitrate by quantize the smoothed bandwidth.

2.2 Actual implementation

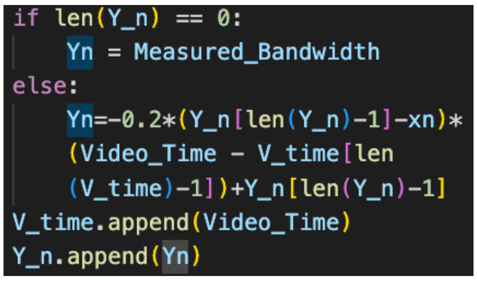
In my implementation, the first step is to get the available bitrate the next chunk can use. According to the previous given example code we can use "R_i" to store the available bitrate. “R_i” is a list which elements are pairs of theoretical bitrate and actual available bitrate. R_i[i][0] is the string form of theoretical bitrate. In the experiment are 500000, 1000000 and 5000000. Additionally, R_i[i][1] is the integer form of actual bitrate which depends on the Internet condition of that time.

At the same time, I use a variable named “xn” to be the estimated bandwidth. According to the algorithm of conventional approach the estimated bandwidth is equal to the previous time throughput. Therefore, the estimated bandwidth can directly get from the input.

The second step is to smooth the estimated bandwidth. I used two methods here: linear regression and EWMA smoother.

First, linear regression. This kind of smooth bandwidth is just for trying. I used all previous estimated bandwidth and chunk number to got a line and used the value of corresponding chunk number to get smoothed bandwidth. However, the problem of this kind of smooth is that if there is a huge drop of the throughput in recent several chunk, the line would give a relatively low value of smoothed bandwidth because considering those low point. Therefore, it may cause the bad performance in some cases.

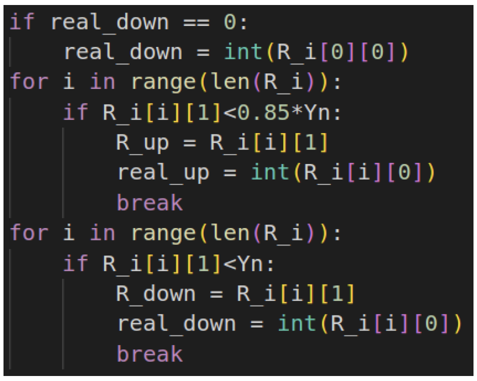
Second, EWMA smoother. According to the equation given by the paper, the EWMA has a better performance after conducting those experiment. EWMA smoother utilizes the relationship of previous smoothed bandwidth, current estimated bandwidth and the time between current download operation and previous one to get a smoothed bandwidth.



```
if len(Y_n) == 0:
    Yn = Measured_Bandwidth
else:
    Yn=-0.2*(Y_n[len(Y_n)-1]-xn)*
    (Video_Time - V_time[len
    (V_time)-1])+Y_n[len(Y_n)-1]
V_time.append(Video_Time)
Y_n.append(Yn)
```

Figure 1: Smooth the estimated bandwidth

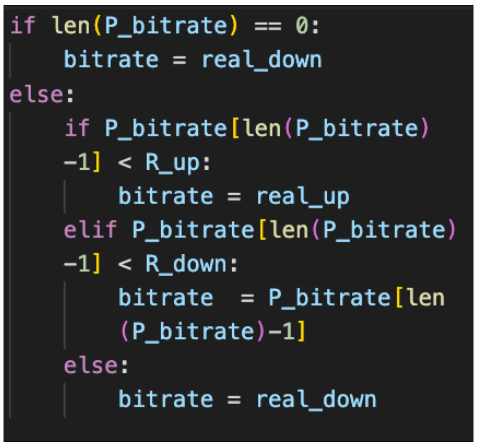
The third step is to quantize the smoothed bandwidth. In order to quantize the smoothed bandwidth, we should first get two boundary. The first boundary is named “R_up” which is the actual bitrate of low boundary which is the maximu value of bitrate less than 0.85 smoothed bandwidth. 0.85 is default value which is given by the paper. The second boundary is named “R_down” which is the actual bitrate of high boundary which is the maximu value of bitrate less than smoothed bandwidth.



```
if real_down == 0:
    real_down = int(R_i[0][0])
for i in range(len(R_i)):
    if R_i[i][1]<0.85*Yn:
        R_up = R_i[i][1]
        real_up = int(R_i[i][0])
        break
for i in range(len(R_i)):
    if R_i[i][1]<Yn:
        R_down = R_i[i][1]
        real_down = int(R_i[i][0])
        break
```

Figure 2: Dead-zone Quantizer step 1

Next step is using these two boundary and the previous bitrate to choose bitrate for next chunk. If the previous bitrate is smaller than "R_up" then the bitrate is equal to corresponding theoretical bitrate of "R_up". If the bitrate is greater or equal to "R_up" and smaller than "R_down", then the bitrate is equal to previous bitrate. Otherwise, the birtate is equal to corresponding theoretical bitrate of "R_down".



```
if len(P_bitrate) == 0:
    bitrate = real_down
else:
    if P_bitrate[len(P_bitrate)
    -1] < R_up:
        bitrate = real_up
    elif P_bitrate[len(P_bitrate)
    -1] < R_down:
        bitrate  = P_bitrate[len
        (P_bitrate)-1]
    else:
        bitrate = real_down
```

Figure 3: Dead-zone Quantizer step 2

3 Result Analysis

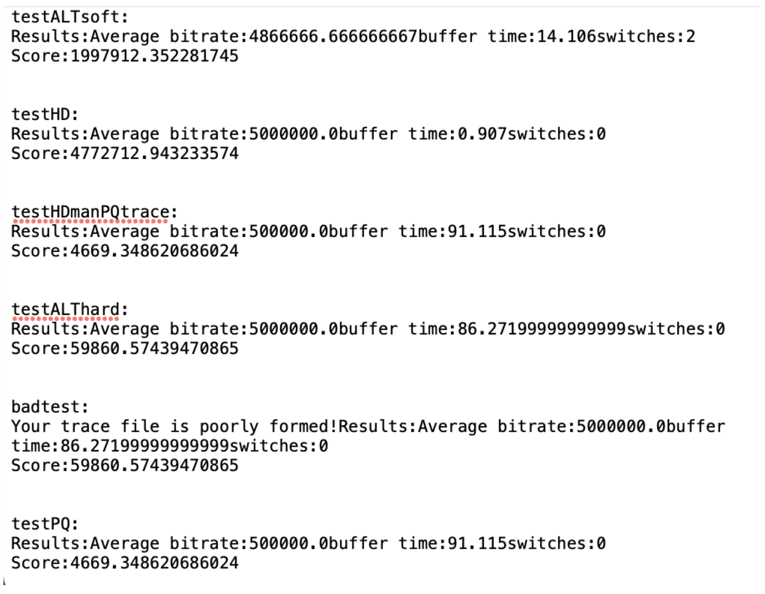
Compared with given example, my implementation algorithm has higher score on first two experiments and equal score of the third and last experiments. However, the given example has better performance on the fourth and fifth experiments. After doing the experiments, I found that the theoretical birtate and actual available bitrate has huge gap. Therefore, it would be better to check this kind of gap in order to get better performance. After I implement this method, all experiments have better or equal performance than the given example.

4 Evaluation

The performance of this algorithm has its own strength on estimating the next round bandwidth, which can help to choose next bitrate for user side server and control both download time and video quality. However, when the actual available bitrate has big gap with the theoretical bitrate, the performance of this algorithm is not good enough. After adding the comparison of smoothed estimated bandwidth and the theoretical bitrate, the performance increases. Additionally, the smooth function may also have optimization approach.

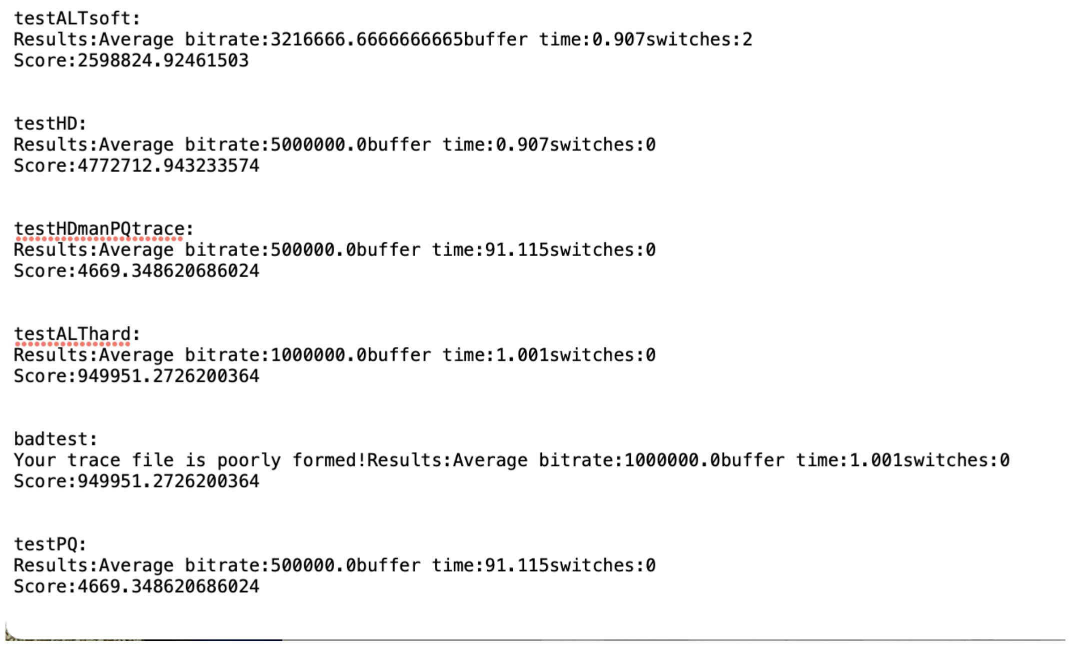
5 Appendix

Additionally, I do not want my code used in the future or public, thank you in advanced. And the following two figures show the grade result of two versions of my implementation.



```
testALTsoft:
Results:Average bitrate:4866666.666666667buffer time:14.106switches:2
Score:1997912.352281745


testHD:
Results:Average bitrate:5000000.0buffer time:0.907switches:0
Score:4772712.943233574


testHDmanPQtrace:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024


testALThard:
Results:Average bitrate:5000000.0buffer time:86.27199999999999switches:0
Score:59860.57439470865


badtest:
Your trace file is poorly formed!Results:Average bitrate:5000000.0buffer
time:86.27199999999999switches:0
Score:59860.57439470865


testPQ:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024
```

Figure 4: Test result without optimization



```
testALTsoft:
Results:Average bitrate:3216666.6666666665buffer time:0.907switches:2
Score:2598824.92461503


testHD:
Results:Average bitrate:5000000.0buffer time:0.907switches:0
Score:4772712.943233574


testHDmanPQtrace:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024


testALThard:
Results:Average bitrate:1000000.0buffer time:1.001switches:0
Score:949951.2726200364


badtest:
Your trace file is poorly formed!Results:Average bitrate:1000000.0buffer time:1.001switches:0
Score:949951.2726200364


testPQ:
Results:Average bitrate:500000.0buffer time:91.115switches:0
Score:4669.348620686024
```

Figure 5: Test result with optimization