Design of an algorithm to predict the download rates

Varsha Thirumakil
Department of Computer Science
Assignment-5
vt2000@wildcats.unh.edu

Abstract— Predictive analytics deals with extracting information from current data and using it to predict the future behavior patterns of the networks. These analysis would play a vital role in several networking techniques. In this paper, we propose an algorithm framework that based on the transfer rates of previous segments determines the maximum request rate that guarantees the next segment will be downloaded with 95% confidence. This has been achieved by carefully designing an algorithm that predicts the confidence level with which the next segment will be downloaded. The algorithm gives us a download rate of prediction which can be compared with the actual rate taken by the request and hence the approximate confidence interval is determined. We evaluate the correctness of the algorithm by analyzing the actual and predicted download rates.

Keywords— download rate, RTT , confidence inerval

I. INTRODUCTION

[1] HLS is a simple and elegant architecture for delivering adaptive bit rate streams to iOS devices and compatible browsers, essentially Safari. [2] Adaptive bitrate video delivery is a combination of server and client software that detects a client's **bandwidth capacity** and adjusts the quality of the video stream between multiple bitrates and resolutions. HLS encodes the original video into multiple variants at various resolutions and bitrates. It then divides each variant into multiple segments. All these are uploaded to a standard HTTP web server. The **goal** here is to make a prediction algorithm for the download rate of the next segment and then determine the probability of success of the download rates.

II. BACKGROUND

Research shows that some basic performance tests conducted to see how **node.js** server behaves compared to Apache proves that node is fast. Really fast. [3] Much faster than Apache -
many more requests per second, higher transfer rate with much smaller number of failed requests at the same time. Hence the node.js server code is used in the experiment.

III. ALGORITHM DESIGN AND ANALYSIS

A. Experiment motivation :

The main motivation of the experiment is to come up with an algorithm that helps the HTTP Live Streaming client to predict the download rate of the next segment based on the previous segment with 95% confidence.

B. Measures :

Initially, the performance of http over ipv4 is analyzed and the **download rates** for the next segments are predicted based on the previous segment values. The **round trip time or the latency** of the request- response interaction is calculated in the experiment which is utilized in designing the prediction algorithm.

C.Algorithm description :

The Date object of the javascript is used to work with time. Date objects are created with new Date(). Using **JS Date** we can find out the start time when the request was sent and the time when the response is delivered.

Hence the round trip time is found. The algorithm works in the simple javascript **client** in the following way:
Step 1: The actual request took time for 1Mb file is found.
Step 2: Then the actual download rate is found using :
transferRate in Mbps = bytesTransferred / (curTime – startTime).
bytesTransfered is 1mb of text or video/audio file.

Step 3: The expected download time is found using the **mean of the previous round trip times and adding the scaling factor of 3.2 to obtain 90% confidence** of the next download rate. Then the download rate in Mbps is found using Step 2.
I also tried finding the mean absolute error(mean i+1 – mean i)
and the obtained results was with 90% confidence.
Step 4: The **setInterval()** method calls a function or evaluates an expression at specified intervals (in milliseconds).
This is used to determine the actual and expected download rates for **desired number of times** and hence get a efficient and precise prediction analysis.
Step 6: If the actual rate is **greater** than the predicted rate most of the times then the prediction algorithm is designed efficiently.
Step 7: To determine the success we should make sure our algorithm does not fail more than 2% of the times among the total downloads.
The algorithm I came up works with 90 – 92 % confidence. As mentioned when testing the algorithm with Ipv6 gives a consistent performance with download rate being ~ 60 Mbps.
To be more sure with the results I also tried giving the maximum priority to the most recent round trip time then taking the mean using the sum of weights and then **adding the scale factor** to it. The download rate hence found also resulted with 90% - 92% confidence.

Standardized results :
To be more standardized with the results **mean absolute error(MAE)** can be calculated.
Once the mean absolute error is found this value can be added with the next prediction to find the results with 90% confidence.

IV. RESULTS AND DISCUSSIONS

Given a server and a client the basic experiment works by client requesting 1MB of text/mp4 from the listening server. The server responds back to the client with the data using the callback function. The algorithm is implemented across rb1/rb2.
The actual rate was greater than the predicted rate for around ~75-80 downloads out of 100 total downloads. From this we can say that algorithm produces the prediction rates with 90 % confidence level and utilization being 0.9 * 100 = 90% (ratio of actual download times and the predictions).

CONCLUSION

Thus an algorithm was implemented with http over Ipv4 on RB1/RB2 that predicts the download rate with which the next segment will be downloaded. To make it simple, analyzing my results produced that among 100 total downloads approximately 80 of the downloads resulted with actual rate more than prediction rate.
Download rate results for http over Ipv4 :
For a text file : ~50 Mbps - 70 Mbps (for http over ipv6 consistently ~60 Mbps – 63 Mbps was produced with very less fluctuations).
For a mp4 file : ~15 Mbps – 19 Mbps
Success of the algorithm worked **with 90% confidence.**

REFERENCES

[1]http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/How-to-Encode-Video-for-HLS-Delivery-95251.aspx

[2]http://www.encoding.com/http-live-streaming-hls/

[3]http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php