Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 06, June-2021

An Implementation for a Prediction based Internet Load Balancing Algorithm

Shaik Aftaab Zia[1], Samarth Segu[2], Varun V Devadiga[3], Vikas T Shankar[4], and Vani K A[5]
Department of Information Science, Dayananda Sagar College of Engineering

Abstract— **Internet load balancing algorithms can be categorised into static and dynamic algorithms. Static algorithms like Round Robin and IP hash are rule based and do not take into account dynamic information like load on individual servers. Dynamic algorithms like Least connections take this into account and aim to distribute traffic more optimally, but lead to requirement of monitors or polling mechanisms to obtain this information. Predictive load balancing algorithms aim to remove this requirement by trying to predict load induced on servers due to requests rather than measuring it directly. We aim to provide an improved implementation of algorithm described by Patil *et al.*[1] and compare this implementation with a static algorithm like Round Robin in terms of performance and resource utilisation. This implementation is for a web application which does text-to-speech synthesis.**

Keywords— **load balancing, predictive load balancing, text-to-speech service**

I. Introduction

Load balancing is an important aspect of any distributed system. If a single server is used for request processing, it raises the issue of creating a single point of failure. With the help of load balancer, traffic can be spread across a cluster of nodes, leading to a better availability and system performance.

A. Static algorithms

Static algorithms distribute network traffic based on a set of predefined rules. They are suitable for systems where most requests require similar amounts of computation. These algorithms have good performance and are easy to scale as cluster size increases.

The drawback with static algorithms is that they do not take into account the current system state, i.e. the amount of resources being used by each node, due to which they don't provide optimal resource utilisation for systems where requests vary in computational requirement.

Popular static load balancing algorithms include the Round Robin, IP hash, Power of two choices and Randomized load balancing.

B. Dynamic algorithms

Dynamic load balancing algorithms follow a greedy approach of distributing load to the server with minimum load. There are various approaches to measuring load on individual servers like using the number of open connections(Least connections algorithms), or Resource based algorithms which attach monitors onto individual worker nodes from which metrics related to each node can be pushed or pulled from the load balancer.

The drawback with the Least connections algorithms is that it does not always distribute traffic to the minimum load server. For instance, a servers A and B are currently processing requests X and Y number of requests, with $X < Y$. However server A may be processing more computationally intensive requests than B so despite having lesser connections server A will have more load. With Resource based algorithms, we need to install monitor processes on worker nodes to measure resource usage like CPU, RAM and disk usage and either push these metrics to the load balancer or have them polled by the load balancer at regular intervals. This leads to additional overhead on the system.

C. Prediction based Dynamic load balancing

The main idea behind prediction based dynamic load balancing is to eliminate the need for monitor processes, but still get the same resource utilisation as resource based algorithms. Instead of measuring system resource usage directly, we try to predict a value X which is linearly proportional to the net system load. Some algorithms also try to predict individual resource usage like IO, CPU and memory usage. This value X(or set of values $X_1, X_2, ..X_n$) is used to determine which server the request has to distributed to.

II. Related work

Most of our work in this paper builds on top the work done by Patil *et al.*[1]. However, there is immense research related to the area of Predictive load balancing and some of the related papers are highlighted in this section.

Patil *et al.*[1] designed a predictive load balancing algorithm for the use case of a Text-to-Speech synthesis application. Their system follows a Master-Slave architecture where the load balancer breaks text input from each request into individual lines and each line is sent to the worker node with minimum current load. Queues are maintained in the Master node for each server in the pool and are populated with values $t_1, t_2, ..t_n$ which correspond to the predicted time values to synthesize lines $l_1, l_2, ..l_n$, which are the lines currently queued for synthesis in that particular worker node. Load on each server is calculated by summing up all values in the server's queue.

Chandra *et al.*[2] developed a predictive algorithm which predicts a request's CPU, IO, and memory requirement and distributes it to the server with minimum load for that particular metric. Using this approach, they were able to reduce the response time of the system. Similar research work was done by Goswami *et al.*[3].

Oikawa *et al.*[4] described an approach for using Machine learning to select the load balancing algorithm based on the current performance metrics. They described a static phase where data is collected and manually labelled to build a training set and a classifier is trained on to it so that the best algorithm for the current load can be predicted.

Thakor *et al.*[5] proposed a prediction based algorithm and compared its performance of predictive load balancing against a static algorithm called Priority load balancing(where each incoming request has a priority associated with it and is directed to the server which maps to that priority value). Predictive algorithms were shown to have better cluster utilisation.

III. Proposed system

Our proposed system follows the same Master-Slave design as described by Patil *et al.*[1]. However, we have developed a general equation for predicting synthesis time for each server by taking into account information like number of CPU cores, RAM and GPU availability of each server rather than specific linear regression models trained for each server to the pool as described in [1]. We have also compared performance of various predictive models for predicting synthesis time.

A. System Design

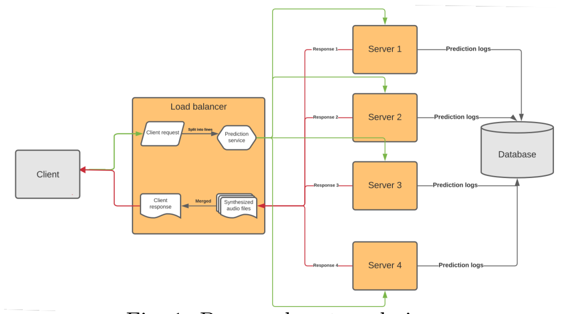

Fig. 1: Proposed system design

Our proposed design is given by figure 1. Text paragraphs from HTTP requests sent to the load balancer(the master node) are split into sentences. For each sentence in the request, the current minimum load server is selected and the sentence is then sent into the prediction service for predicting the time taken by the selected server for synthesizing it. Once a worker node synthesizes a particular sentence, it logs the time taken to a database. This data can be used for retraining the model used by the prediction service and serves as a way of continuously improving the prediction service.

The actual request distribution process is described by Algorithm 1.

Algorithm 1 Predictive Load balancing

```
1:  global variables
2:      $sLoad$                        ▷ Hashtable for server-load
3:      $sAddrs$                         ▷ List of Server IPs
4:  end global variables
5:  procedure ProcessRequest($request$)
6:      $text$    $request.text$
7:      $procList$    []
8:      $splitLines$    $text.split(‘.’)$
9:      for line in splitLines do
10:         $minServer$    $sAddrs[0]$
11:         $minload$    INF
12:         for addr in sAddrs do
13:             $load$    SUM($sLoad[addr].values()$)
14:             if $load < minload$ then
15:                 $minload$    $load$
16:                 $minServer$    $addr$
17:         $predTime$    PREDICT($addr, line$)
18:         $sLoad[addr]$    $sLoad[addr] + predTime$
19:         $pid$    ASYNC PROXYREQ($line, minServer$)
20:         $procList.add(pid)$
21:     $responses$    AWAITALL($procList$)
22:     $resultWav$    MERGE($responses$)
23:     return RESPONSE($resultWav$)
24: procedure Predict($addr, line$)
25:     $serverConfig$    DB.$getConfig(addr)$
26:     $cpu$    $serverConfig.cpu\_cores$
27:     $ram$    $serverConfig.total\_ram$
28:     $gpu$    $serverConfig.has\_gpu$
29:     $nword$    LEN($line.split(‘.’)$)
30:     $nchar$    LEN($line$)
31:     return MODEL.$predict(cpu, ram, gpu,$
            $nword, nchar)$
```

We chose the Lasso Regression model for predicting synthesis time as it provided a lower error rate on our dataset than LinearRegression(chosen by the authors of [1]). The dataset was generated manually using the PredictionLogs table described earlier. It consisted of 232 samples, 80% of which were used for training and rest for testing.

For synthesizing speech from text, we've used the FastSpeech model[6]. We eliminated the WaveGlow model in the FastSpeech due to performance constraints.

IV. Results

The results designed in this section were obtained under the assumption that worker nodes and the load balancer are connected over a very high speed network , sending and receiving data over which is negligible. They were obtained by using 3 worker nodes with the following configurations:

ID	RAM(GB)	CPU Cores	GPU Availability
1	1.5	2	No
2	1	1	No
3	1	1	No

Each of the above nodes is a Docker(a containerization software) container with limitations set to CPU and Memory. This method is similar to using Virtual Machines(VM) and was chosen as it has less performance overhead than a VM.

A. *Predictive Model*

We've trained the following models on the generated dataset(232 samples, 80% for training and 20% for testing) and have obtained the following results

Tabla I: Test error for predictive model

Model	Hyper Parameters	MSE
Linear regression	-	12.883
Random Forest	estimators=100	14.859
GBDT	learning_rate=0.1	13.354
Ridge Regression	alpha=125	12.612
Lasso Regression	alpha=0.125	**12.473**

Other hyper-parameters were set to their default values in the Scikit-Learn Python Package.

Compared against the Mean Squared Error(MSE) metric, linear models performed the best(Lasso, Ridge and Linear Regression). This can be expected as the relationship between the features chosen is linear.

B. *Comparison with Round Robin load balancing*

The following are average request response times of the application using Round Robin load balancing and Predictive load balancing for a 331 word, 1829 character paragraph over 9 measurements.

Algorithm	Response time(seconds)
Round Robin	258.42
Predictive load balancing	**256.17**

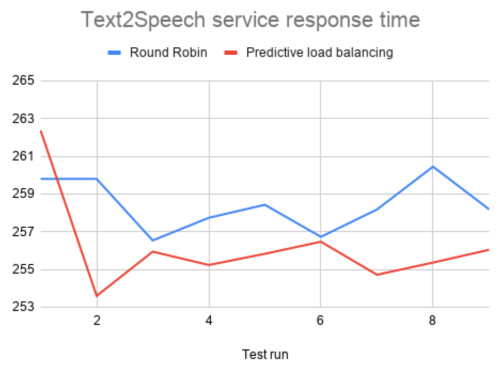

Fig. 2: Test measurement vs Average response time

The above results have some variance due to small variance in the synthesizing speech through the FastSpeech model, but generally Predictive load balancing was found to perform better than Round Robin load balancing for this use-case application.

V. Conclusion

Predictive internet load balancing is not widely seen in most applications as most applications do not require optimal resource utilisation and static algorithms like Round Robin and IP hash work just fine. However, for some specific use cases, dynamic load balancing is important and prediction based algorithms can help in overcoming the drawbacks associated with them.

The following is the implementation for the work described in this paper - `https://github.com/Aftaab99/Text2SpeechService`.

Referencias

[1] Ganesh Patil and Santosh Deshpande, "Prediction based dynamic load balancing system for text to speech conversion," 11 2019, pp. 1122–1127.
[2] P. Chandra and Bibhudatta Sahoo, "Prediction based dynamic load balancing techniques in heterogeneous clusters," 01 2008.
[3] Gil-Haeng Lee, Heung-Kyu Lee, and Jung-Wan Cho, "A prediction-based adaptive location policy for distributed load balancing," *Journal of Systems Architecture*, vol. 42, no. 1, pp. 1–18, 1996.
[4] C. Oikawa, Vinicius Freitas, Marcio Castro, and Laércio Lima Pilla, "Adaptive load balancing based on machine learning for iterative parallel applications," 03 2020, pp. 94–101.
[5] C. Oikawa, Vinicius Freitas, Marcio Castro, and Laércio Lima Pilla, "Adaptive load balancing based on machine learning for iterative parallel applications," 03 2020, pp. 94–101.
[6] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu, "Fastspeech: Fast, robust and controllable text to speech," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. 2019, vol. 32, Curran Associates, Inc.