A Solution for Instant Response of Cloud Platform
Based on Nginx + Keepalived

Min HAN¹, Deng-guo YAO²,* and Xiao-lin YU³

¹School of Information Science and Technology, Southwest Jiaotong University,
Sichuan Chengdu 610031, China

²School of Software Engineering, Chengdu University of Information Technology,
Sichuan Chengdu 610225, China

³Chengdu Westone Information Industry Inc, Sichuan Chengdu 610041, China

*Corresponding author

Keywords: Cloud platform, High concurrency, Load balancing strategy, Virtual IP, Nginx, Keepalived.

Abstract. How to fulfill load balancing in high concurrency status and thereby to achieve instant response of cloud platform is always a focus of attention to the industry. This document presents an instant response solution for cloud platform based on Nginx + Keepalived, which, by means of Nginx, enables selection of proper load balancing algorithm based on the scenario requested by client, and detection of server working status by using Keepalived to solve single point of failure. This solution is implemented fully with open-source software, and will allow high-concurrency requests of tens of thousand per second ensure good user experience and achieve the goal of low cost and high availability, making it particularly suitable for small-/medium-sized cloud platforms with economic applicability requirements. Test results show that, with respect to analysis on 50% or 80% of the requests, the average response time for 10,000 concurrent requests is only 101 ms and 152 ms, with memory usage up to 22% and CPU peak power up to 80%, which meets the expected response time shorter than 3 seconds in high-concurrency scenario of tens of thousand visits per second.

Introduction

With rapid development of Internet+ and cloud computing technology, people's daily life is increasingly dependent on various services of cloud platforms. However, how to carry out load balancing and achieve instant response of cloud platform under high-concurrency condition is a spotlight in the industry. Commonly used load balancing components include hardware and software. With respect to small-/medium-sized cloud platforms, due to financial and material constraints, hardware load balancer is expensive and cannot meet their requirements for economic applicability. Therefore, they prefer to use open-source software to achieve load balancing [1].

Open-source load balancing software which is currently used commonly mainly includes LVS (Linux Virtual Server), Haproxy, Seesaw, Nginx and so on. Nginx has the advantages of low resource consumption, high concurrency, open source code and easy expansion, so it is used by many small-/medium-sized cloud platforms as a load balancing component on the server side to enable back-end Web servers to deal with user requests in a high-concurrency environment [2]. Ilias Marinos et al. [3] optimized Nginx-based load balancing server based on Netmap framework and by using network stack memory model, achieving higher throughput and lower CPU utilization. David [4] et al. analyzed the reverse proxy advantages of Nginx and proposed a load balancing optimization algorithm with real-time feedback ability, which improved the connection speed for task processing. Handoko Hero et al. [5] used Haproxy and Keepalived to make load balancer perform better. Kim Taewoo et al. [6] studied the scalability and availability of web-based medical expert system. In order to meet the maximum available services of the Web system, they developed a load balancing structure using Nginx to solve the problem of server traffic and improve the service availability. To ensure
service quality and offer users good game experience during concurrent connection of a large number of users in social network games, Song-Min Kim et al. [7] used Node.js + Nginx technology to construct game engine server, achieving the goal of connecting more users in shorter response time. Based on the viewpoints and research results of the aforesaid papers, this document proposes an instant response solution based on Nginx + Keepalived cloud platform, which, by means of Nginx, enables selection of proper load balancing algorithm based on the scenario requested by client, and detection of server working status by using Keepalived to solve single point of failure. It has been proved that this solution achieves the goal of low cost and high availability, and can support high concurrency scenarios with tens of thousands of visits per second, making it particularly suitable for small and medium-sized cloud platforms with economic applicability requirements.

**Demand Analysis**

The cloud platform studied in this document is a typical small and medium-sized Web application system, where ten nodes are virtualized to act as Web servers in the back end, without requirement for cross-room deployment. Load balancing components on the platform should meet the following requirements:

1. When the concurrent traffic is not less than 10,000/s, the average response time of the platform should shorter than 3 seconds.
2. Cost efficiency, can support virtual host, allowing concurrent visits of tens of thousands in ordinary hardware environment.
3. Working on the seven-layer network protocol, having the ability of setting multiple strategies to conduct corresponding streaming strategy and algorithm for the application.
4. Be highly reliable, automatically switches to another normal server to respond in case that one node fails in internal network.
5. Have good scalability, all network devices are plug-and-play, making it possible to dynamically increase or decrease operations on cluster scale.
6. Have good stability, be capable of conducting load balancing operations even when network condition become harsh or most of ports are closed.
7. Installed and configured simply, convenient for testing.

**Key Technology**

**Nginx Load Balancing Technology**

Load balancing is essentially a distribution strategy for client requests, reasonably allocating pressure to the server so that one server is not overloaded while the other servers are idle. Here the "reasonably" means that a proper load balancing algorithm will be chosen based on server status.

Nginx is a high-performance lightweight HTTP and reverse proxy server compiled by Igor Sysoev, a Russian. It is capable of responding to up to 50,000 concurrent connections with very low consumption of memory, CPU and other system resources while the system is operating stably [8]. Currently, Nginx has been running on Rambler Media - the largest portal in Russia - for many years, and more than 20% of the virtual host platforms in Russia are using Nginx as reverse proxy server. In China, many websites such as Baidu, Sina, Douban, Netease, Tencent and Xunlei have used Nginx as Web server or reverse proxy server [9].

The load balancing technology used by Nginx is VIP (Virtual IP)-based HTTP load balancing technology, working in the "application layer", also known as "seven-layer load balancing", which includes the following functions:

1. Reverse proxy
   It means that, user does not directly access the back-end Web server, but sends the request to Nginx, which then forwards the request to the back-end Web server. Similarly, after the back-end
Web server finishes processing, the result will be sent to Nginx first, and then Nginx sends it to the client.

The reverse proxy configuration includes four steps: Set host header; Set true IP address of the client; Disable cache; Set reverse proxy address.

(2) Nginx buffering mechanism

Nginx can temporarily store the processing results from the back-end Web server in response to the client's access request, thereby closing the connection between Nginx and the back-end early and reducing IO loss.

(3) Load balancing strategy and algorithm

The server allocation strategy provided by Nginx load balancing component include four parts: Round-Robin algorithm, least-connected algorithm, IP_Hash algorithm, and Weighted algorithm, and is also compatible with two third-party algorithms: Fair algorithm and URL_Hash algorithm. Comparisons between these six algorithms is listed in Table 1 below.

**Table 1. Comparison of Six Load Balancing Algorithms.**

<table>
<thead>
<tr>
<th>Load Balancing Strategy</th>
<th>Features and Advantages</th>
<th>Existing Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-Robin</td>
<td>Client requests are sent in chronological order to the back-end Web server, selecting in turn a currently available server in a circular fashion. Since the algorithm does not record the current connection status, the response time is very fast, making it especially suitable for server performance and load condition all with average level.</td>
<td>Real servers are different in performance and load, so this algorithm may cause one server to go down quickly due to poor performance and high load, while another server with good performance is running at low load.</td>
</tr>
<tr>
<td>Least-Connected</td>
<td>Each time, a server with the lowest number of connections will be found to deal with requests.</td>
<td>However, actually the server with the least connections is not necessarily the optimal one.</td>
</tr>
<tr>
<td>IP_Hash</td>
<td>Requests will be allocated based on the client IP address where the request is sent, and this algorithm will ensure that requests from the same client are distributed to one server, greatly reducing the workload in synchronization of the context.</td>
<td>This algorithm cannot be used separately as a load balancing algorithm, because once the determined back-end server is down, this algorithm will fail.</td>
</tr>
<tr>
<td>Weighted</td>
<td>The servers will be weighted differently based on individual difference between them, so that the weight can be used to artificially control the number of requests to be processed by a server and thereby host resources will be used effectively.</td>
<td>In real world, the health status and load of a server often change dynamically, so it is difficult to set a reasonable weight.</td>
</tr>
<tr>
<td>Fair</td>
<td>Requests are allocated based on the response time of the back-end server, with priority given to those with shortest response time.</td>
<td>Intelligent calculation has to be conducted based on page size and load time, making the algorithm expensive and occupy larger server resources. A third-party algorithm for which upstream_fair must be installed.</td>
</tr>
<tr>
<td>URL_Hash</td>
<td>Requests are allocated based on the URL to be visited, ensuring that requests for the same URL are all responded to by the same back-end server. This algorithm can further improve efficiency of back-end cache server.</td>
<td>A third-party algorithm for which a Hash package for Nginx must be installed. Similar with IP-Hash, this algorithm cannot be used separately as a load balancing algorithm, because once the determined back-end server is down, this algorithm will fail.</td>
</tr>
</tbody>
</table>

To sum up, each algorithm has its own Features and is applicable to a certain scenario. In actual application, proper strategy may be chosen based on different scenarios such as the amount of user visits, user's means of visit, performance and scale of server cluster, so as to apply service strategy more reasonably.
Keepalived Technology

Keepalived is a highly reliable component specially developed in Linux system to realize Virtual Router Redundancy Protocol (VRRP) \[10\]. Keepalived consists of one primary server and several backup servers, and the same service configuration is deployed on both primary server and backup servers, providing services by using a single virtual IP address that automatically drifts to the backup server when the primary server fails.

1. High availability achieved by VRRP

VRRP provides external services with a virtual IP, and the virtual router contains at least two or more physical routers which work together. At any time only one router acts as Master, and the MAC address of its network adapter is bound to that virtual IP, while any other nodes act as Backup, which do not provide external network function. All Backups only receive the notification of VRRP status from the Master to monitor the operation status of the Master. When the Master fails and Backups cannot receive the message from the Master. Then a Backup with the highest priority will be chosen with election algorithm, and its MAC address will be bound to the external virtual IP to make it the new Master, which will continue to provide external services and ensure continuous availability of services.

2. Detection of server working status

Backups will receive a packet sent by the Master at given time through heartbeat detection of Keepalived. If the Master fails to respond for a given time, it will be deemed to have failed and removed from the system, and a Backup will be used in place of it. Once the damaged server returns to work properly, Keepalived will automatically add it to the server cluster without any manual intervention, and all that has to be done manually is to repair the faulty server \[11\].

Instant Response Solution Based on Nginx + Keepalived

Design of Load Balancing Network Framework

Based on demand analysis, a load balancing network framework is proposed as shown in Figure 1, which consists of user and administrator layer, load balancing layer, and back-end Web server layer.

![Figure 1. Load Balancing Network Framework.](image)

(1) User and administrator layer

This layer receives user requests and forwards them through virtual routes to the proxy scheduling server which currently provides external service on load balancing layer.
(2) Load balancing layer

This layer is in an architecture of Nginx + Keepalived and contains a group of proxy scheduling servers, realizing VRRP through Keepalived and providing a unified virtual IP for users and administrator. These proxy service scheduling servers work in master-slave mode, and different Nginx load balancing strategies are designed based on different application scenarios. Requests from client are reasonably allocated to the back-end Web server. At the same time, Keepalived is used to detect working status of these proxy dispatching servers, avoiding single point of failure in them.

(3) Back-end Web server layer

Ten nodes are virtualized to act as Web server on this layer. After calculation by load balancing layer, the selected Web server is responsible for dealing with user requests and feeding the result data back to users and administrator through proxy scheduling server.

**Design of Load Balancing Strategy Based on Nginx**

When the default Round-Robin algorithm is used for load balancing, it may be the case that, some nodes are already in high load, but requests to be dealt with still grow at high speed, while the other nodes in low load remain idle due to inconsistent physical performance of servers, and different complexity of user requests. Therefore, based on different user application scenarios, this solution chooses different load balancing distribution strategies to make full use of back-end Web servers, so as to achieve instant response on cloud platform.

(1) With respect to continuous requests from the client, IP_Hash algorithm will be used to allocate requests to the Web server that the user or administrator visited last time. Thus, the context is all in current server, the quickest response will be achieved. Problems caused by session sharing can be avoided at the same time.

(2) For requests for the same URL resource, URL_Hash algorithm will be used as much as possible to allocate the requests to the same Web server. In this way, the caching mechanism of server will be effectively utilized, and it is not necessary for each server to cache the same resources and occupy cache space repeatedly.

(3) As a Supplement, weighted algorithm will be used in addition to IP_Hash and URL_Hash algorithms.

**Definition 1:** Define \( Wi_{\text{performance}} = \{W1_{\text{performance}}, W2_{\text{performance}}, ..., Wi_{\text{performance}}, ..., Wk_{\text{performance}} \} \), \( 1 \leq i \leq k \), which is weighted performance of the ith server. It is decided based on Indicators of physical server, such as number of CPU cores (\( N_i_{\text{CPU}} \)), dominant frequency of CPU (\( F_i_{\text{CPU}} \)), size of memory (\( S_i_{\text{mem}} \)), rate of memory (\( R_i_{\text{mem}} \)), type of disk (\( T_i_{\text{disk}} \)), rotating speed of hard disk (\( R_i_{\text{disk}} \)), and space of disk (\( S_i_{\text{disk}} \)), which are static indicators of a Web server.

**Definition 2:** Define \( Wi_{\text{feature}} = \{W1_{\text{feature}}, W2_{\text{feature}}, ..., Wi_{\text{feature}}, ..., Wk_{\text{feature}} \} \), \( 1 \leq i \leq k \), which is weighted processing capacity of the ith server. It is decided by the factors collected based on current working status of Web server such as utilization of CPU (\( U_i_{\text{CPU}} \)), utilization of IO (\( U_i_{\text{IO}} \)) and utilization of memory (\( U_i_{\text{mem}} \)), which are dynamic indicators of a Web server.

To sum up, in design of weight for Web server, two factors - weighted performance and weighted processing capacity of Web servers, should be taken into consideration, which is a typical problem of fuzzy comprehensive evaluation, as shown in Figure 2. This scheme uses the classical Analytic Hierarchy Process (AHP) for calculation.
Status Detection and Fault Isolation Based on Keepalived

Three proxy scheduling server nodes are designed in this solution, among which Node 1 serves as Master, Node 2 and Node 3 serve as Backup. VRRP is realized by Keepalived to provide a unified virtual IP for external service, which is bound to the MAC address of Node 1. Node 2 and Node 3 receive VRRP status message from Node 1 through Keepalived, and monitor whether Node 1 works properly. When Node 1 fails or downs, Keepalived will choose one with higher priority from Node 2 and Node 3 by using election algorithm, bind its MAC address with the unified virtual IP, and act as Master to provide proper external services.

Keepalived works mainly in network layer, transport layer and application layer. In the network layer, Keepalived sends an ICMP packet to each node in the server cluster through Internet Control Message Protocol (ICMP). If it does not receive a returned response packet, Keepalived will think that the node is faulty, and report the fault and delete the node. In the transport layer, Keepalived detects whether port of the server cluster node is faulty through TCP port connection. If it does not detect the response data on the detected port, Keepalived will think that the node is faulty and delete the node. In the application layer, Keepalived detects status of the server based on additional user-customized scripts. If the detection result is inconsistent with user's settings, the server will be deemed to have failed and removed from the server cluster.

Tests

In this document, Webbench, a website stress testing tool, is used for stress testing and simple data analysis.

1. Webbench -c 100 -t 60 http://192.168.122.1/pay/pno1001?type=01&userid=xxx

Performance of the whole application is tested by respectively simulating requests from different clients within 60 seconds and via interface of the front-end server requesting load balancing. The test results are given in Table 2.
Table 2. Test Results.

<table>
<thead>
<tr>
<th>Nginx Test Indicator</th>
<th>Concurrent Requests</th>
<th>100</th>
<th>500</th>
<th>1,000</th>
<th>5,000</th>
<th>10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>≈25</td>
<td>≈18</td>
<td>≈20</td>
<td>≈21</td>
<td>≈23</td>
</tr>
<tr>
<td></td>
<td>100 pages/sec</td>
<td>≈8%</td>
<td>≈12%</td>
<td>≈15%</td>
<td>≈18%</td>
<td>≈22%</td>
</tr>
<tr>
<td></td>
<td>Memory Usage</td>
<td>32%</td>
<td>41%</td>
<td>47%</td>
<td>66%</td>
<td>80%</td>
</tr>
<tr>
<td></td>
<td>CPU Peak Power</td>
<td>≈16</td>
<td>≈17</td>
<td>≈18</td>
<td>≈19</td>
<td>≈20</td>
</tr>
<tr>
<td></td>
<td>100KB/sec</td>
<td>28ms</td>
<td>34ms</td>
<td>46ms</td>
<td>70ms</td>
<td>101ms</td>
</tr>
<tr>
<td></td>
<td>Response Time to 50% of Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>50%</td>
<td>35ms</td>
<td>36ms</td>
<td>50ms</td>
<td>88ms</td>
</tr>
<tr>
<td></td>
<td>Response Time to 80% of Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>50%</td>
<td>36ms</td>
<td>36ms</td>
<td>50ms</td>
<td>88ms</td>
</tr>
</tbody>
</table>

Based on the test results given above, with respect to analysis on 50% or 80% of the requests, the average response time for 10,000 concurrent requests is only 101 ms and 152 ms; the maximum memory usage is 22% and CPU peak power is 80%, which meets the expected response time shorter than 3 seconds in high-concurrency scenario of tens of thousand visits per second. Therefore, this solution perfectly meets the design requirements and achieves the goal of low cost and high availability.

Summary

This document, aiming at the principle of economic applicability for small-/medium-sized cloud platforms, and from the perspective of open-source software, presents an instant response solution for cloud platform based on Nginx + Keepalived, which, by means of Nginx, enables selection of proper load balancing algorithm based on the scenario requested by client, and detection of server working status by using Keepalived to solve single point of failure, achieving the goal of low cost and high availability, allowing high-concurrency requests of tens of thousand per second, ensuring good user experience, and thereby providing a practicable solution for small-/medium-sized cloud platforms.

Acknowledgements

This research was financially supported by the National Key R&D Program of China (2017YFB1401403) and the Basic Applied Research Projects in Sichuan Province (2017JY0044).

References


