# Study on Probability Statistics of Unbalanced Cloud Load Scheduling

Shuo-yu Zeng(✉), Yu-jun Niu, and Hong-e Wu

School of Mathematics and Statistics, Nanyang Institute of Technology, Nanyang 473000, China
wuhonghe321313@163.com

**Abstract.** Aiming at the problem of unstable equilibrium probability in modern load scheduling applications, a statistical method of unbalanced probability in cloud load scheduling is proposed. The weights and anti-saturation factors are calculated, the servers are grouped, the fuzzy cyclic iterative control of dynamic network resources is realized, and the network packet cloud load scheduling is designed. By comparing with the common methods, it is proved that the method designed in this paper can guarantee high equilibrium probability and good stability in a certain program.

**Keywords:** Cloud load · Scheduling imbalance · Probability statistics

## 1 Introduction

In recent years, the rapid growth of network applications, such as the rise of mobile Internet, cloud computing, big data, and other services, has a profound impact on people's daily life and brought great convenience to users. Since the 1970s, the demand for the network is only a simple end-to-end transmission. IP data packet mainly contains the network address of the source host and the destination host. Almost all the traffic on the Internet is based on the TCP/IP architecture. At that time, TCP/IP can also meet this demand well. Since the opening of the network, the network equipment is constantly updated, and the network traffic that can be handled is also growing. However, there is no breakthrough in the structure of the network. Today's network environment is complex, and various applications emerge in endlessly, such as the development of the cloud. Its computing, storage, service, and security are the most popular projects at present. Behind these rising commercial projects are various data centers and tens of thousands of server clusters. These switches and servers provide the most basic support for the upper layer services, while the data center is the bridge in the whole network operation [1]. With the growth of business and the continuous expansion of the data center, the computing power of the server is thousands of times higher than that of the original computer. In the process of data transmission between the server and the server, the link plays a role of connection. The whole data center, the network link is a complex topology. Today, the link utilization rate of the large-scale network center is only 30%–40%, which is just to deal with sudden large flow; The utilization of the whole link bandwidth is low. In addition to the extensive demand for


S. Liu and L. Xia (Eds.): ADHIP 2020, LNICST 348, pp. 223–231, 2021.
https://doi.org/10.1007/978-3-030-67874-6_21

the Internet, the traditional data center has not been able to fully support the bandwidth demand of all kinds of application processing, so the link transmission of the data center has become the bottleneck of cloud computing. The research of network link load balancing can greatly improve the network utilization, and at the same time, it can make the network accept more traffic requests and improve the network throughput, which will be more beneficial to the network data interaction and user experience.

The research of network load balancing has always been a hot spot in network research. It can guarantee the performance of the network and improve the user experience; The calculation of the optimal path is the basis of network interconnection [2]. At this time, the emergence of software-defined networking (SDN) can adapt to the current network requirements. Centralized control, separation of control and data, network programmability and monitoring capability of the whole network have made breakthroughs in network load balancing. Statistical method of unbalance probability in cloud load scheduling. The weights and anti saturation factors are calculated, the servers are grouped, the fuzzy cyclic iterative control of dynamic network resources is realized, and the network packet cloud load scheduling is designed.

## 2 Probability Statistics Method of Unbalanced Load Scheduling in Network Packet Cloud

### 2.1 Cloud Server Load Calculation

It overcomes the problem that static algorithm can not solve the load of cloud server, which leads to the imbalance of cluster. The controller should dynamically collect the load information of the server, including the utilization rate of the central processing unit (CPU), memory utilization rate, and network bandwidth utilization rate [3]. Represented by $C_i$, $M_i$, and $B_i$ ($i$ represents the $i$-th server), the load of the $i$-th server is:

$$L_i = \varepsilon_1 * C_i + \varepsilon_2 * M_i + \varepsilon_3 * B_i \tag{1}$$

Among them, $\sum_i \varepsilon_i = 1$; $\varepsilon_i$ represents the impact factor. A larger value indicates that this performance parameter has a greater impact on this server. The size of $\varepsilon_i$ can be statically set according to the business that the balanced service faces. To let the equalizer better understand the service capabilities of each service, the capacity of each server is collected here:

$$R_i = \mu_1 * F_i + \mu_2 * N_i * 1000 + \mu_3 * MT_i + \mu_4 * BT_i \tag{2}$$

Among them, $\sum_i \mu_i = 1$; $\mu_i$ larger value indicates that this parameter accounts for a larger proportion of the capacity of the computing server. The size of $\mu_i$ can be set according to the different services targeted by the balanced service. $F_i$ is the CPU frequency; $N_i$ is the number of CPU cores; $MT$ is the total memory; $BT_i$ is the total bandwidth.

According to the defined load $L_i$ and capacity $R_i$, the weight of each server is obtained:

$$W_i = 1/\left((1 - L_i) * \sqrt{R_i}\right) \tag{3}$$

## 2.2 Anti-saturation Factor Calculation

When the number of sent requests reaches a certain value, the number of server response requests suddenly drops to zero and continues for some time. A measure taken by system software to prevent crashes, also known as "denial of service" or "fake death" [4]. When the server is "fake dead", the process is that the response time slowly increases, then jitters, and then increases sharply. Once it increases sharply, it is called to enter the saturation state. If an anti-saturation factor is added during the jitter, the server weight The value is falsely increased, and the threshold is used to prevent it from entering the saturation state, to effectively prevent the server from entering the saturation state and affecting the services provided.

$$WS_i = \begin{cases} \frac{1}{(1-L_i)*\sqrt{R_i}}, no - time - shake \\ \frac{1}{(1-L_i)*\sqrt{R_i}} + \varsigma_i, time - shake \end{cases} \tag{4}$$

Here, the resource utilization of the cloud host exceeds the threshold and the higher packet entered by the cloud host is determined as the response time jitter, plus the anti-saturation factor; otherwise it is determined that the response time is not jitter [5].

The anti-saturation factor $\varsigma_i$ is the result of multiplying the cloud host weight by the static scale. Because the configuration of each cloud host is different, the weights will be different and $\varsigma_i$ will also be different. When the load information of the cloud host is collected, first, the resource utilization of the cloud host will be judged whether it exceeds the threshold. If it exceeds, the weight will be set to 1, and the cloud host will enter the high load group. If it is not exceeded, it is determined whether the weight of the cloud host enters a group with a higher weight. If the weight of a cloud host is entered, the cloud host's weight will increase by $\varsigma_i$. After the cloud host weight is increased by $\varsigma_i$, the cloud host will make the average weight of the group to be higher, so that the probability that the group will be allocated more requests is smaller, thereby achieving the purpose of anti-saturation.

## 2.3 Cloud Server Grouping

The anti saturation factor can prevent the cloud server with large weight from entering the saturation state, so it will not significantly increase the service response time. However, for servers with low weight, if the equalizer does not update its load information in time, the equalizer will continue to distribute the load, resulting in a rapid increase of its load, which may lead to overload. Unbalance the entire cluster [6, 7]. To solve this problem, servers with similar weights are grouped into a group and a

distribution request is issued in that group instead of being distributed to a server. According to the weight, the server is divided into five groups, as shown in Table 1.

**Table 1.** Grouping servers by weight

| Group | Weights | Group | Weights |
|---|---|---|---|
| 1 | (0, 0.05] | 4 | (0.5, 0.75] |
| 2 | (0.05, 0.25] | 5 | (0.75, 1] |
| 3 | (0.25, 0.5] | | |

From the first group to the fifth group, the server will process more and more requests, and the processing time will be longer and longer [8]. Here, according to the different weights, the time for the equalizer in the controller to collect the servers in the packet will be different. The servers of the first group and the second group do not change much in weight in a certain period of time due to the small number of requests they process [9, 10]; In the same way, the fourth group and the fifth group need more time to process more requests, and the weight will not change greatly in a certain period of time, so the time interval for collecting the load information of these two groups can be larger; But for the third group of servers, which are in the transition stage between the second group and the fourth group, they are very sensitive to the weight information [11]. Once the weight changes, the group will not accept the request again, so the collection interval for the weight information of servers in this group will be shorter. Set five groups of weight information collection intervals as follows: $\tau_3 < \tau_2 = \tau_4 < \tau_1 = \tau_5$. In addition, when the virtual machine in the host pool enters the fourth and fifth groups due to too many requested resources, because of the existence of the anti-saturation factor, it can ensure that the virtual machine in the two groups is in the pseudo saturation state, and can still process the accepted requests normally. If there is a new request coming at this time, the request will not be sent to the back end, and the SDN controller will be triggered to inform the cloud platform to establish a new virtual machine instance to join the host pool. This will be reflected in the next research.

### 2.4 Fuzzy Cyclic Iterative Control of Dynamic Network Resources

In the cloud computing environment, firstly, the fuzzy cyclic iterative control algorithm is used to extract the dynamic network resource balance scheduling characteristic parameters [12, 13]. Compared with the recursive algorithm and the VC++ standard library function nth element, it shows that the algorithm is more efficient and reliable than the traditional recursive algorithm. Compared with the standard library function nth element, it has obvious advantages in time efficiency. Establish the dynamic network resource weight distribution mechanism [14, 15], calculate the orthogonal weighted constraint equilibrium ratio of network resources, and carry out the dynamic network resource fuzzy cyclic iterative control. The specific process is as follows:

Suppose that $\{x_n\}_{n=1}^{N}$ represents the collection of dynamic network resources in the cloud computing environment. According to the following vector control methods, a new resource mapping relationship $x_n$ is constructed in the stack space. $k$ represents the sampling point time series of dynamic network resource load in the cloud computing environment. $\tau$ represents the dynamic network time delay parameter.

Then formula (5) can be used to represent a time span of dynamic network resource operation:

$$h(\tau_i, t) = \sum_{i=1}^{N_m} a_i(t) e^{\phi} \varsigma(t - (\tau_i, t)) \tag{5}$$

Where, $a_i(t)$ represents the number of tasks currently running, $e^{\phi}$ represents that the physical machine is in saturated running state, $(\tau_i, t)$ represents that the server is not currently running, $\varsigma(t - (\tau_i, t))$ represents the fastest progress of resource task execution, and $N_m$ represents the waiting time of virtual machine resources during task execution. Suppose that R represents the trust relationship function of network resources, establishes a dynamic network resource weight allocation strategy, calculates the orthogonal weighted constraint equilibrium ratio $e_{ij}$ of network resources, and evaluates the attribute weight of network resources classification. The effective ratio function of dynamic network resource tasks in the cloud computing environment is given as follows:

$$E(i,j) = \begin{cases} \big(e_{ij} - e(i,j)\big)/(e_{\max} - e(i,j)), e(i,j) < e_{ij} \\ \big(e_{ij} - e(i,j)\big)/(e(i,j) - e_{\min}), e(i,j)) \geq e_{ij} \end{cases} \tag{6}$$

In the formula, $e(i,j)$ represents the attribute weight of multi-source heterogeneous resources, $e_{\max}$ represents the highest efficiency of the task, and $e_{\min}$ represents the lowest efficiency of the task. For a point on the dynamic network resource set a in the cloud computing environment, The equilibrium probability calculation parameter is:

$$R'' = \varpi_1 C_i + \varpi_2 D_i + \varpi_3 M_i + \varpi_4 N_i \tag{7}$$

Where, $N_i \in N_m$, $\varpi_\eta (\eta = 1, \ldots, 4)$ represents the weight coefficient, $C_i$ represents the dynamic network resource weighted constraint equilibrium ratio, $D_i$ represents the time parameter, and $M_i$ represents the time sampling period in the cloud computing environment.

Suppose that $h(A, B)$ represents the space directed fuzzy distance between resource set A and B, and $N_A$ represents the covariance vector of data information flow of dynamic network resource set A. The delay of information collection caused by the approximate calculation of data flow equilibrium scheduling is $T$, and the fuzzy cyclic iterative control is used to control the dynamic network resource scheduling process. Set a set of control parameters $G(V, E)$ and $\sin k \in V$, find a method of network resource scheduling set to minimize the dynamic network load parameter $TS(u)$. Define the dynamic network load balancing scheduling parameters, select the corresponding delay response parameters, and use Eq. (8) to calculate:

$$Y = \frac{h(A,B)}{N_A \times U_i} + D_i \frac{[\varsigma_{ik}(t) + \Theta(t)]}{k} \tag{8}$$

Where, $\varsigma_{ik}(t)$ represents the information of the dynamic network control node, $U_i$ represents the dynamic network resource classification attribute, $k$ represents the network load intensity, and $\Theta(t)$ represents the trust value obtained by the dynamic network resource search at the latest time [10]. According to the construction of data information flow, the time scale characteristic parameter $f_i(t)$ of dynamic network load resource is extracted, and the request information of dynamic network load resource scheduling task with scale $c$ is $p_c$. The calculation formula (9) of unbalanced probability of cloud load scheduling is:

$$S_{\varpi} = \sum_{a=1}^{c} p \frac{1}{T} \sum_{k=1}^{T} [U_i \cdot f_i(t)] \tag{9}$$

In formula (9), $a$ represents the number of network edge nodes, and $\frac{1}{T}$ represents the dynamic network node quota. So far, the unbalanced probability calculation of cloud load scheduling is completed.

## 3 Method Test Experiment

In order to verify the performance of this method, a comparative experiment is designed with the common probabilistic methods. Through comparison, the hypothesis of the experiment is verified.

### 3.1 Experimental Program

In order to make the designed scheme and algorithm have universality and scalability, the collection of load information and the processing of load by the load balancer are implemented in a Java modular manner. The collection of each kind of load information only needs to implement an interface and generate the corresponding class. The load balancer handles the load in the same way. Versatility is reflected in the fact that the classes currently implemented are the most basic load information collected, and factors that should be considered for load balancing; scalability is reflected in interfaces and classes. After the server receives the request, it consumes the server's CPU, memory, and bandwidth resources through the script. Each requested service will result in corresponding resource consumption. For example, the CPU will consume 10%–20% of the consumption. The specific CPU consumption can be changed Script tuning, and after a period of time, the load generated by each request will automatically exit after completion. The Linux performance monitoring tool dstat is used in the experiment to record each cloud host server at regular intervals, and the recorded results will be saved in a log. Each record will have time, which is convenient for later comparison when analyzing data The load of each cloud host. The network bandwidth test uses the iperf tool, and the CPU and memory tests are performed using scripts. First, use the

resource utilization of each cloud host to compare and analyze various strategies. As time changes, observe the impact of each strategy on the cluster equilibrium and the impact of each strategy on the cluster equilibrium at the same time. Then use the standard deviation of resource utilization of the cloud host cluster to compare different strategies, and experiment with the effects of different strategies on the overall cluster balance. The main comparisons are three strategies: Random, RoundRobin, and ASGS. Experimental result analysis the experimental environment configuration is shown in Table 2. During the test of the three strategies, the same software and hardware were used, but configuration changes were made at the equalizer through the RESTful structure according to the different strategies tested.

**Table 2.** Experimental environment configuration

| Physical machine & VM | Role | CPU core | Frequency/GB | Internal storage/GB | Drive |
|---|---|---|---|---|---|
| compute1 | OpenStack compute1 | 4 | 3.2 | 64 | e1000e |
| controller | OpenStack controller | 4 | 3.2 | 64 | e1000e |
| compute1_vm | Server 101, 103 | 1 | 3.2 | 16 | virtio-pci |
| compute1_vm | Client 106 | 2 | 3.2 | 8 | virtio-pci |
| compute1_vm | Server 107, 108 | 2 | 3.2 | 8 | virtio-pci |

### 3.2 Result Analysis

In order to compare the equilibrium effects of the three strategies, two indicators are used for measurement: the first is to compare the resource utilization of each cloud host in the cloud host cluster at each moment. The closer the resource utilization of each cloud host under the three strategies at each moment is, the higher the balance is.

As shown in Fig. 1, H1 to H4 represent four cloud hosts, which represent three strategies (Random (1), Round Robin (2), ASGS (3) strategy) The resource utilization of each cloud host at each moment. For example, at the first moment, under the Random strategy, the resource utilization of the fourth cloud host is much higher than the other three. This results in a severe imbalance in the cluster load at the first moment, and the balancing effect is not very good as the moment changes. In contrast, Round-Robin and ASGS do not produce this situation, and the balance of ASGS is significantly better than that of Random and Round Robin after 300 times. Therefore, the order of the equalization effect is getting better and better: ASGS. The second is to compare the standard deviation of resource utilization of each host at each time, so as to measure the balance moment of each virtual machine at this time. If the standard deviation is close to 0, the higher the balance. The experimental results in Fig. 1 show that the ASGS method also has a good balance. It shows that the proposed method can effectively improve the stability of cloud load scheduling.

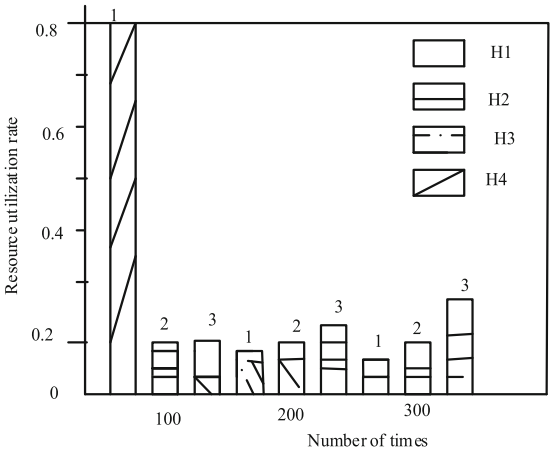


**Fig. 1.** Comparison of cloud host resource utilization in three strategies

## 4 Conclusion

Because of the unstable equilibrium probability of load scheduling in the modern application process, a statistical method for the unbalanced probability of cloud load scheduling is designed. Through the contrast experiment with the common methods, it is proved that the method designed in this paper can guarantee high equilibrium probability and good stability in a certain program.

## References

1. Any fruit. Improving laser network load balance technology in cloud computing environment. Laser J. **38**(2), 137–141 (2017)
2. Ge, W., Ye, B.: Improved priority schedule scheduling algorithm for load balancing priority. J. Shenyang Univ. Technol. **39**(3), 241–247 (2017)
3. Fu, Y., Liu, B., Shu, Y.: Research on the multi-path load balancing algorithm in the data center of the schubby-tree data center based on SDN. Comput. Appl. Softw. **34**(9), 147–152 (2017)
4. Qu, H., Zhao, J., Fan, B., et al.: Application of ant colony optimization in software definition networks. J. Beijing Univ. Posts Telecommun. **40**(3), 51–55 (2017)
5. Fu, M., Wu, F., Huang, F., et al.: Research on performance of multipath routing algorithm based on software definition network. J. Fuzhou Univ. (Nat. Sci. Edn.) **45**(5), 628–634 (2017)
6. Parkhom, A.: virtual machine scheduling method based on load balancing in cloud computing environment. Inf. Comput. (Theoret. Edn.) **44**(21), 48–50 (2017)
7. Zhong, B., Jiang, L.: Distributed network load balancing strategy using IMKVS combined with NFV in software defined networks. Comput. Appl. Res. **36**(5), 1504–1509 (2019)
8. Xin, Z., Lei, T.: Simulation of dynamic network resource scheduling in cloud computing environment. Comput. Simul. **34**(12), 402–406 (2017)

9. Liu, S., Li, Z., Zhang, Y., et al.: Introduction of key problems in long-distance learning and training. Mob. Netw. Appl. **24**(1), 1–4 (2019)
10. Razzaghzadeh, S., Navin, A.H., Rahmani, A.M., et al.: Load balancing based on statistical model in expert cloud. Majlesi J. Electr. Eng. **13**(4), 61–71 (2019)
11. Fu, W., Liu, S., Srivastava, G.: Optimization of big data scheduling in social networks. Entropy **21**(9), 902 (2019)
12. Madni, S.H.H., Latiff, M.S.A., Ali, J.: Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. Cluster Comput. **22** (1), 301–334 (2019)
13. Pan, Z., Liu, S., Sangaiah, A.K., et al.: Visual attention feature (VAF): a novel strategy for visual tracking based on cloud platform in intelligent surveillance systems. J. Parallel Distrib. Comput. **120**, 182–194 (2018)
14. Chaudhary, D., Kumar, B.: Cloudy GSA for load scheduling in cloud computing. Appl. Soft Comput. **71**, 861–871 (2018)
15. Aruna, M., Bhanu, D., Karthik, S.: An improved load balanced metaheuristic scheduling in cloud. Cluster Comput. **22**(5), 10873–10881 (2019)