Design and Implementation of a TCP Long Connection Load Balancing Algorithm Based on Negative Feedback Mechanism

WEI Chunlei\textsuperscript{1,2,a}, HOU Jixin\textsuperscript{1,2,b}, MA Dechao\textsuperscript{1,2,c}, ZHAO Jinghu\textsuperscript{1,2,d} and SUN Yunfeng\textsuperscript{1,2,e}

\textsuperscript{1}NARI Group Corporation/State Grid Electric Power Research Institute;
\textsuperscript{2}NARI Technology Development Co. Ltd.;
\textsuperscript{a}weichunlei@sgepri.sgcc.com.cn, \textsuperscript{b}houjixin@sgepri.sgcc.com.cn, \textsuperscript{c}madechao@sgepri.sgcc.com.cn, \textsuperscript{d}zhaojinghu@sgepri.sgcc.com.cn, \textsuperscript{e}sunyunfeng@sgepri.sgcc.com.cn

Abstract. Dynamic load balancing is an important way to improve the resource utilization and parallel computing performance of multi-server system. For the mass of terminals under the Internet of Things scenario high frequency periodic escalation and receiving messages scene, the terminal equipment and message server need to establish a TCP long connection to communicate, the accumulation of messages different degree of weight will cause the load tilt. In this paper, an algorithm based on the socket buffer feedback mechanism is proposed, which takes the server load (CPU, memory, connection rate and socket buffer cache) as the load indicator, collects the load indicator through the load agent and calculates the discrete coefficient of the server socket buffer. The dispatch controller calculates the weight of each server, and the higher the weight, the lighter the server load. The experiments show that the CMTS more realistically feedbacks the server load status than the least connections. As the number of cluster server connections and message sending rate increase, the average cluster response time is shorter.

1. Introduction
Now we have officially entered the 5G commercial era, lots of devices connected to the network will generate massive data after being sensed, and the data processing becomes more and more heavy. The Internet of things system is also divided into cloud, pipe, edge and the end. The cloud is responsible for processing data, and the pipeline is responsible for transmission, while the edge is responsible for access to the terminal.

Constrained by CPU, memory, network and other resources. The number of TCP connections that long connection service can bear is limited. The server cluster and load balancing technology provide a solution for this\cite{1}. Li Qiang, et al. Studied and designed the load balancing system of mass long connection\cite{2}.

Terminals in the field of the Internet of things report messages periodically in high frequency, and network traffic happens periodically like waves\cite{3}. The size, frequency, quality of service, bandwidth and resource utilization of the server will affect the performance of the server when the terminal and the message server establish a long TCP connection communication. Therefore, it is of great significance to select the appropriate load index\cite{4}. The load index is a standard for measuring the load status of the cluster system. The improvement of cluster performance depends mainly on the...
selected load index[5]. Tang Dan et al.[6] show that taking the function of CPU and various I / O request as the load index better than using only CPU[7].

This paper proposes an algorithm based on feedback mechanism, which uses centralized control and load broker to collect load information of cluster nodes and allocate tasks. The CMTS algorithm introduces the amount of data accumulated in the socket buffer as a key indicator, taking into account factors such as CPU performance, CPU average load, memory utilization, and cluster online connection rate. It can not only comprehensively measure the node load state, but also has small test cost, which conforms to the ideal load index standard[8]. Experimental results show that the CMTS algorithm reduces cluster delay and improves cluster resource utilization compared to the least connections algorithm.

2. Load balancing and cluster load index parameters

2.1. Load balancing

Server cluster is an effective structure to realize highly scalable and highly available network services[9]. When some nodes in the cluster are overloaded or fail, the load scheduler will dispatch requests to the nodes with lighter load. This process does not require the participation of operation and maintenance personnel to automatically complete the load adjustment.

Load balancing improves the performance of the whole cluster by balancing the load among the cluster nodes. The static load balancing algorithm formulates an allocation strategy based on the server's hardware configuration or the content of the request. It does not require additional information to determine the load, but it has poor adaptability and is not suitable for distributing complex requests.

The dynamic load balancing algorithm[10] obtains the load capacity information of the server cluster in real time, and dynamically distributes the requests to the nodes with lighter load. From the overall cluster performance point of view, the improvement effect is better than the impact of additional overhead on the system.

2.2. Cluster load index parameters

In the actual cluster system, different requests consume different resources. The algorithm[11] considers CPU and memory utilization, but does not consider the actual network traffic. The dynamic algorithm of web server cluster classifies requests by type and considers CPU, memory utilization and network traffic[12], but it is not suitable for TCP long connection scenarios.

This paper proposes a CMTS load balancing algorithm, which uses kernel socket buffer as the core load indicator, combined with three auxiliary indicators of average CPU load, memory utilization, and cluster online connection rate. The load agent collects four load indicators on each node periodically, and calculates the node weights. When a cluster node is allocated for a terminal request, it is allocated based on the node weights.

2.2.1. CPU average load. In Linux, CPU average load refers to the average number of processes that the system is in a running and noninterruptible state in a unit time, that is, the average number of active processes. The average CPU load calculation formula is: LoadAvg = CPUNumber * CPUCore * ThresholdUpLimit (where: the upper limit of the threshold is 75% by default). When the system starts, load the file system /proc provided by the kernel into memory, and the system automatically generates /proc/ loadavg when it starts. This paper obtains the average CPU load by reading /proc/loadavg.

2.2.2. Memory utilization. Memory is mainly used to store system and application instructions, data, cache, etc. Many performance problems are caused by the exchange of operating system due to insufficient memory. Therefore, it is necessary to use some tools to monitor the memory usage. Use the free tool to check the system memory usage. The following is an example of free output:
Total Used Free Shared Buff/Cache Available
Mem: 3049476 299944 2073964 7280 675568 2504704
Swap: 0 0 0

The value is in bytes by default. The above two lines are the usage of physical memory (kernel state) and swap (user state).

Total is the total memory size; Used is the size of used memory, including shared memory; Free is the size of unused memory; Shared is the size of shared memory; Buff/Cache is the size of cache and buffer; Available is the memory available for the latest process. Linux provides a tool in /proc file system to monitor the overall utilization of system memory resources, that is, /proc/meminfo. The calculation formula for memory utilization is as follows:

\[ \text{MemUsage} = 100 \times \frac{\text{MemTotal} - \text{MemFree} - \text{Buffers} - \text{Cached}}{\text{MemTotal}} \]

2.2.3. Cluster online connection rate. Cluster online connection rate refers to the ratio between the number of node connections per unit time and the average cluster connections, calculated as follows:

\[ L(T_i) = T_i \frac{\sum_{i=1}^{N} T_i}{N} \]  

Where T represents the online connections and N represents the number of cluster nodes.

2.2.4. Kernel socket buffer. In Linux system, TCP/IP and general network driver software are the core components of the kernel, while device driver is the add-on module. Packets pass through these kernel components with socket buffer, which is responsible for transferring message data between source and sink. This paper only studies the socket layer of TCP protocol, and the socket buffer whose connection state is estab[13].

This paper obtains the TCP connection status by executing the ‘ss’ command. For example, to monitor the TCP connection status of port 1884, the command is as follows:

```
ss | grep -E 1884 | egrep tcp
```

If you want to monitor multiple ports, the port format is 1884|1885|1886.

3. CMTS algorithm design

The CMTS algorithm framework consists of two parts: The first part is the load agent responsible for collecting the indicators of each server and calculating its weight into the cache module. The second part consists of two modules, the scheduling controller and the load balancer, as shown in the following figure:

![CMTS algorithm framework](image)

**Figure 1.** CMTS algorithm framework.

It can be seen from Figure 1 that the terminal accesses the server cluster through WLAN, and the scheduling control module dynamically adjusts the load index parameters by using the feedback mechanism. The scheduling control module selects the nodes with smaller load according to the load balancer.
3.1. CMTS algorithm theoretical presupposition

3.1.1. Definition. Definition 1: \( kT \) represents \([kT, (k+1)T)\), where \( k>0 \);

Definition 2: \( A(T) \) represents the processing capacity of the server in unit time;

Definition 3: \( A(kT) \) represents the processing capacity of the server in \( kT \) time;

Definition 4: \( S(i, kT) \) represents that the read / write queue capacity of the i socket is not 100% in \( kT \) time;

Definition 5: \( N(kT) \) represents the socket connections allowed to access in \( kT \) time.

So we can get the following formula:

\[
A(kT) = A(T) \cdot kT \geq \sum_{i=1}^{N(kT)} S(i, kT)
\]

(2)

If the above formula is true, the server can also establish a socket connection, otherwise it is not allowed to assign a TCP connection to the machine. Therefore, the selection of \( A \) and \( T \) will affect the efficiency of the system.

(1) Statute. The CMTS algorithm convention is as follows: (1) Continuous time discretization into multiple time intervals: \([0, T), [T, 2T)... [kT, (k+1)T)...\], two consecutive update times are marked as a stage, the end time of stage \( K \) is \((k + 1) T\), \( k > 0 \);

(2) The observation window is a measure of several sampling intervals;

(3) Generally, a server only deploys one service, and the read-write queue stack analysis only targets at several ports of a specific service process;

(4) Statistical analysis of changes in the socket buffer read and write queues in the service process;

(5) The processing power of the server in unit time is expressed by weight coefficient. The stronger the processing power of the server is, the larger it is, and vice versa;

(6) The ability of socket to read / write queue in \( kT \) time is expressed by the following formula:

\[
R(r, kT) = \frac{Q_r[(k+1)T] - Q_r[kT]}{Q_{r_{max}}}
\]

(3)

\[
R(w, kT) = \frac{Q_w[(k+1)T] - Q_w[kT]}{Q_{w_{max}}}
\]

(4)

(7) In this paper, the overall threshold value of the server is 70%, and the single index threshold value of the server is 75%;

(8) The default weight of the server is 1.0

It is represented by server weight, so the calculation becomes the key. The sampling interval \( T \) needs to be given according to the experimental data. If \( T \) is too small, it will increase the burden of the server. If \( T \) is too large, it may not reflect the timeliness of data accumulation on the socket in time.

3.2. CMTS algorithm description

The workflow of the scheduler, load broker, weight calculation and feedback module in the CMTS algorithm is shown in Figure 2. The load broker collects the node load information according to the sampling interval, and periodically reports the load information to the cache. If the node does not respond within the set time interval, the monitor of the scheduling control system considers that the node is not reachable, resets the node to the non working state, returns the weight to zero, and the new connection will not be assigned to the node. If the node has response in the next sampling interval, recalculate the weight of the node.
3.2.1. **Symbol definition.** The definition of the symbols in the algorithm in this paper is shown in Table 1. 0, T, 2T, ..., between two consecutive moments is recorded as a collection interval, and the end time of the interval k is kT.

| Symbol | Definition |
|--------|------------|
| T      | Sampling interval |
| k      | Positive integer such as 1,2,3... |
| N      | Number of servers |
| \( \overline{S} \) | Socket queue mean |
| \( L(S) \) | Server integrated load |
| \( L(C) \) | CPU load |
| \( L(M) \) | Memory load |
| \( L(T) \) | TCP long connection load rate |
| \( L(Q) \) | Load of socket queue |
| \( L(SK) \) | Socket buffer load of server |
| \( \lambda_{cpu} \) | Relative weight coefficient of CPU load |
| \( \lambda_{mem} \) | Relative weight coefficient of memory utilization |
| \( \lambda_{connect} \) | Relative weight coefficient of server connection rate |
| \( \lambda_{socket} \) | Relative weight coefficient of socket buffer |
| W      | Server weight |

3.2.2. **CMTS algorithm derivation.** According to the theory of fuzzy set, if any resource or cluster online connection rate exceeds a certain proportion, the system will no longer be able to bear the additional load.

In this paper, a set of dynamically adjustable coefficients is introduced to represent the weight of each load parameter. Comprehensive load reference calculation formula[14], comprehensive load formula in this paper:
\[
AL(S) = \lambda_{cpu} \cdot L(C) + \lambda_{mem} \cdot L(M) + \lambda_{conn} \cdot L(T) + \lambda_{socket} \cdot L(SK)
\]

(5)

The default values are 0.1, 0.1, 0.2 and 0.6.

\(L(SK)\) is the standard deviation of \(L(Q)\) (the maximum value of read queue and write queue), and \(L(Q)\) is the load of socket queue.

\[
L(Q) = \left\{ \begin{array}{ll}
\sum_{i=1}^{N(1T)} \frac{Q_{kt} - Q_{(k-1)T}}{Q_{max}} & [Q_{kt} \geq Q_{(k-1)T}] \\
1 - \sum_{i=1}^{N(1T)} \frac{Q_{kt} - Q_{(k-1)T}}{Q_{max}} & [Q_{kt} < Q_{(k-1)T}]
\end{array} \right\} \quad (k \geq 1)
\]

(6)

\(Q_{max}\) represents the maximum value of socket queue, \(Q_{kt}\) represents the queue size of socket at the \(k\)-th time interval point, and \(T\) represents the time interval

\[
L(SK) = \left( \frac{\sum_{k=1}^{N(1T)} (L(Q_k) - \bar{Q})^2}{N(kT) - 1} \right)^{\frac{1}{2}}
\]

(7)

If the average value of queues is the same, \(L(SK)\) can reflect the load of socket. If the mean is different, the standard deviation needs to be compared. The socket discrete coefficient is the relative statistic of the discrete degree of socket data, which is mainly used to compare the discrete degree of socket queue. The larger the discrete coefficient is, the larger the data discrete degree is; the smaller the discrete coefficient is, the smaller the data discrete degree is.

\[
CV(SK) = \frac{L(SK)}{Q_{max}}
\]

(8)

\[
L(SK) = \frac{1}{2}[CV(SK_r) + CV(SK_w)]
\]

(9)

Where \(CV(SK_r)\) and \(CV(SK_w)\) are the discrete coefficients of socket read queue and write queue respectively. The formula for calculating the server weight is:

\[
w = w + A \cdot 3 \sqrt{SU - AL(S)}
\]

(10)

Where \(A\) is the dynamic adjustment coefficient and \(SU\) is the system utilization rate, which cannot be 0. \(AL(S)\) represents the comprehensive load of the node.

The administrator sets an initial weight \(w\) for each node before cluster work. With the change of node load, \(w\) is automatically adjusted by the scheduling control system. In order to avoid too much weight, this paper limits the range of weight [\(w, C^*w\)], \(C\) is an adjustable constant and \(C=10\). If \(w\) is not zero, query the load index of the node and calculate the comprehensive load value. In this paper, the following weight calculation formula[15] is introduced to adjust the node weight according to the comprehensive load of the node.

\[
w = \begin{cases} 
w + A \cdot 3 \sqrt{0.95 - AL(S)[AL(S) \neq 0.95]} \\
w[AL(S) = 0.95] \end{cases}
\]

(11)

Where \(A\) is the dynamic adjustment coefficient, which cannot be 0, and \(AL(S)\) is the comprehensive load of the node. In the above weight formula, we assume that 0.95 is the system utilization rate to be achieved, \(A\) is the number of comprehensive load terms, and this article takes \(A = 4\). The following three situations are obtained by formula analysis:

First, it is unchanged when \(AL(S) = 0.95\);
Second, it becomes smaller when $\alpha L(S) > 0.95$;
Third, when $\alpha L(S) < 0.95$, it becomes larger.

If the new weight is greater than $C^* w$, we set the new weight to $C^* w$. If the difference between the new weight and the current weight exceeds a set threshold, the new weight is set in the scheduling parameters to avoid interrupting the scheduling overhead.

3.2.3. CMTS algorithm analysis. In actual use, if the weights of all nodes are greater than $w$, the entire cluster is in an overloaded state, and new nodes need to be added; on the contrary, if the weights of all nodes are lower than $C^* w$, the current system load is light.

The algorithm calculates the comprehensive load according to the average CPU load, memory utilization, cluster online connection rate, and socket buffer, which truly reflects the real-time load of the nodes, establishes a numerical relationship between the comprehensive load and the weight, and selecting the node with the lower load to allocate the request can effectively suppress load skew.

4. Experimental verification and analysis of CMTS algorithm
In order to verify the CMTS algorithm proposed in this paper, a message service cluster was constructed in the experimental environment, and the minimum connection number algorithm on the cluster and the CMTS algorithm proposed in this paper were compared and tested.

Experimental environment is shown in Table 2. Three identically configured message servers, one stress program server and one integrated server equipped with load bokers, scheduling control, caching and load balancers. The stress program simulates the device sending batch messages to the server with the specified connections (100 bytes per message).

Table 2. Server configuration.

| Server type       | Operating system | CPU model                        | CPU cores | RAM(G) | Hard Disk(G) |
|-------------------|------------------|----------------------------------|-----------|--------|--------------|
| Stress server     | Ubuntu 16.04.5 LTS | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz | 2         | 2      | 50           |
| Integrated server | Ubuntu 16.04.5 LTS | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz | 2         | 2      | 50           |
| Message server    | Ubuntu 16.04.5 LTS | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz | 2         | 2      | 50           |

This paper designs two test schemes to verify the rationality of CMTS algorithm design and cluster performance.

Scheme 1: Test the cluster's overall average response time for cluster connections in unsampled and sampled scenarios. This solution keeps reporting 10,000 messages per second when 5,000, 10,000, ... 40,000 connections are established, as shown in Table 3.

Table 3. Average response time (ms).

| Number of connections | Node is not sampled | Node is sampled |
|-----------------------|---------------------|-----------------|
| 5000                  | 28                  | 30              |
| 10000                 | 43                  | 50              |
| 15000                 | 58                  | 67              |
| 20000                 | 83                  | 98              |
| 25000                 | 135                 | 158             |
| 30000                 | 305                 | 358             |
| 35000                 | 400                 | 552             |
From the test results in Figure 3, it can be concluded that the sampling program of the node consumes some computer resources, and the performance improvement is less when the number of connections is small. Compared with the unsampled machines, the cluster response time is relatively extended.

Scheme 2: Test the rationality of the scheduling algorithm. The scheduling controller uses the minimum connection number algorithm and CMTS to calculate the simulator to simulate the user to send a connection to the message server within a specified period of time, as shown in Table 4.

**Table 4. Average response time (ms).**

| Number of connections | Least connections algorithm | CMTS algorithm |
|-----------------------|-----------------------------|----------------|
| 5000                  | 50                          | 40             |
| 10000                 | 90                          | 50             |
| 15000                 | 150                         | 100            |
| 20000                 | 189                         | 150            |
| 25000                 | 237                         | 205            |
| 30000                 | 346                         | 240            |
| 35000                 | 500                         | 450            |

**Figure 3.** Cluster average response time under unsampled and sampled scenarios.

**Figure 4.** Cluster average response time under Least connections, CMTS algorithm.
As shown in Figure 4, the message sending rate is 30,000 messages per second. However, as the number of terminal requests for connections increases, the overall average response time of the CMTS algorithm is significantly lower than the minimum connection number algorithm. And when the number of connections is 32,000, the CMTS algorithm is more ideal for node load balancing. In the process of continuous testing for 8 hours, the number of connections on one node in the cluster is far less than the other two nodes. The connections requested by the terminal are all allocated to the nodes with smaller connections, and the stress test program increases. The message sending rate is 60,000 messages per second. After the cluster runs for 2 hours, the node has been in an overload state, and the other two nodes are relatively idle. Compared with the minimum connection number algorithm, the CMTS algorithm provides a more realistic feedback on the load status of the cluster, and the nodes in the cluster are more reasonably utilized, which reduces the average response time of the cluster and improves the utilization of cluster resources.

5. Concluding remarks
The CMTS algorithm calculates the comprehensive load according to the four indicators of the nodes, calculates the node weights, and distributes the terminal requests to the nodes with lighter loads. Theoretical analysis and experiments show that: the nodes with few connections have socket buffer accumulation, and the nodes with a lot of connections may be idle. If the minimum connection number scheduling algorithm is used, the connections will be allocated to the cluster nodes with socket buffer accumulation. Obviously does not meet the original intention of load balancing. The CMTS algorithm well balances the load of each node in the cluster, reduces the average response time, and improves the overall concurrency of the cluster.

The algorithm in this paper has a certain delay, which mainly means that the cluster load parameters are counted after a sampling period. The sampling period needs to be obtained by stress testing before the cluster is deployed. It cannot be too small, and too small will cause additional overhead to the cluster; too large will not truly reflect the current load status of the cluster. This paper has certain limitations: First, the algorithm in this paper is only compared with the minimum connection number algorithm in the average response time. In the future, it will be compared in multiple dimensions and indicators in terms of algorithm speed and complexity. Second, the algorithm passes the evaluation of the experimental environment, and more complex and unpredictable factors will appear in the real environment. Third, the algorithm in this paper does not consider how to perform load migration schemes for nodes with heavy cluster loads. Therefore, the practicability and versatility of the algorithm will be tested and verified in a real cluster environment in the future.

Acknowledgment
This work is supported by State Grid Corporation China project(Nari Group's independent project, software development and application of the Power IoT Management Center)

References
[1] Li Jie Research on dynamic load balancing algorithm based on LVS cluster [D]. Nanchang University of Aeronautics, 2018
[2] Li Qiang, Wang Fengqin Soft load balancing method and system supporting massive long connections [P]. China Patent: 106254377, 2016-12-21
[3] Ma Qiang Design and implementation of OpenFlow-based server cluster dynamic load balancing architecture in a virtualized environment [D]. Lanzhou: Lanzhou University, 2014
[4] Yu Dunfu, Li Hongjian, Tang Hong, et al. Research on dynamic load balancing algorithm based on feedback mechanism [J]. (Application Research of Computers, 2012, 29 vol 2) pp 527-529
[5] Brancokri.JC, Ordonezdem. Load indices-past,present and future[C] // Proc of International Conference on Hybrid Information Technology, WashingtonDC: IEEE. Computer Society, 2006:206-214
[6] Tang Dan, Jin Hai, Zhang Yongkun Performance Evaluation of Cluster Dynamic Load Balancing System [J]. Chinese Journal of Computers, 2004, 27 (6): 803-811
[7] KUNZ, T. The Influence of different workload descriptions on a heuristic load balancing scheme[J]. (IEEE Trans on Software Engineering, 1991, 17 vol 7) pp 725-730
[8] ROTH P C, ARNOLD D C, and MILLER B P. MRNet A software-based multicast/reduction network for scalable tools[C]//Proc of ACM/IEEE Conference on Supercomputing. New York: ACM, 2003: 21.
[9] Zhang Wensong Linux server cluster system (1) [EB/OL]. https://www.ibm.com/developerworks/cn/linux/cluster/lvs/part1/index.html, 2002
[10] Gong Mei, Wang Peng, Wu Yue A transparent dynamic feedback load balancing algorithm for cluster systems [J]. (Journal of computer applications, 2007, 27 vol 11) pp 2662-2665
[11] Yang Mingji, Wang He, Zhao Jiafeng. Research on Load Balancing Algorithm Based on CPU and Memory Utilization [J]. (Science and Technology Bulletin, 2016, 32 vol 4) pp 160-164
[12] Zhang Lin, Li Xiao-ping and Su Yuan. A Content-based Dynamic Load-Balancing Algorithm for Heterogeneous Web Server Cluster[J]. ComSIS, 2010, 7 vol 1 pp 153-162
[13] Anatomy of M. Jones. Linux Network Stack-From Socket to Device Driver [EB / OL]. https://www.ibm.com/developerworks/cn/linux/l-linux-networking-stack/index.html, 2010 -9-20
[14] Zhou Songquan An Improved Cluster Dynamic Load Balancing Algorithm [J]. Computer and Modernization, 2012, 36 vol 1 pp 135-139
[15] He Long, Dai Xinfu, Wu Huafeng, et al. A load balancing algorithm based on CPU load averaging [J]. Computer and Digital Engineering, 2012, 40 vol 8 pp 22-24