I.J. Modern Education and Computer Science, **2017, 12, 57-65**
Published Online December 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2017.12.07

A Dynamic Feedback-based Load Balancing Methodology

Xin ZHANG, Jinli LI*, Xin FENG
School of Computer Science and Technology
Changchun University of Science and Technology
Email: qianxiaweiguang@126.com

Received: 08 October 2017; Accepted: 08 November 2017; Published: 08 December 2017

***Abstract*—**With the recent growth of Internet-based application services, the concurrent accessing requests arriving at the particular servers offering application services are growing significantly. It is one of the critical strategies that employing load balancing to cope with the massive concurrent accessing requests and improve the access performance is. To build up a better online service to users, load balancing solutions achieve to deal with the massive incoming concurrent requests in parallel through assigning and scheduling the work executed by the members within one server cluster. In this paper, we propose a dynamic feedback-based load balancing methodology. The method analyzes the real-time load and response status of each single cluster member through periodically collecting its work condition information to evaluate the current load pressure by comparing the learned load balancing performance with the preset threshold. In this way, since the load arriving at the cluster could be distributed dynamically with the optimized manner, the load balancing performance could thus be maintained so that the service throughput capacity would correspondingly be improved and the response delay to service requests would be reduced. The proposed result is contributed to strengthening the concurrent access capacity of server clusters. According to the experiment report, the overall performance of server system employing the proposed solution is better.

***Index Terms*—**Load balancing, dynamic feedback, server cluster, distributed computing

I. Introduction

In recent years, various Internet-based application services have gradually been employed in all the areas of human life. Through the response procedures, there are a considerable number of service requests arriving at the server within a short time and demanding to be answered with less delay[1]. Therefore, it is the server throughput that would directly affect the quality of application services. It directly reflects the parallel processing capability of servers (clusters) that support service capabilities. The higher parallel processing capability means that, within a fixed period, the number of service requests to be dealt with can be relatively larger[2]. As a solution to improve the parallel processing capability, the load balancing technology groups a large number of services requests with multi-point distribution to averagely assign the requests to each of the member servers (according to their responsibilities in the cluster, we respectively categorize the member servers as "master control node" and "working node") within the same cluster[3]. Thus each of the working servers only needs to cope with a similar number of the assigned requests, and then the server cluster is enabled to provide the parallel processing capability[4]. The access efficiency of service requests is thus improved. In this paper, we propose a load balancing method based on dynamic feedback. By dynamic feedback mechanism, we collect the working conditions of each working node in real time, to dynamically adjust the workload distribution scheme according to operating conditions during service request access, so that each working node Undertake similar workloads to the extent feasible and avoid unbalanced load distribution. The method can efficiently adapt to dynamic operating conditions during concurrent processing and has good real-time capability and flexibility.

II. Related Work

The earliest balancing mechanism is performed by employing DNS (Domain Name System) which configures a host with multiple mapping addresses through DNS configuration[5]. After the interactions with the host from the arrived user service requests, the requests are redirected by the host to the different mapping addresses according to a preset principle[6]. The servers at the mapping addresses would deal with the requests in parallel, i.e., the preliminary load balancing operations.

However, in this way, the load balancing efficiency is relatively low and the working performance is limited by the network topology. The load balancing doesn't gain much progress.

According to the research of Nikolaou et al., the actual resource utilization rate of working nodes is the core factor to achieve a better load balancing performance[7]. One of the main objectives of the load balancing scheme

is to rationally plan the workload distribution and equalize the resources of all working nodes. Cardellini and Bryhni et al. compared and studied a variety of load balancing methods, which suggests the advantages and disadvantages of load balancing strategies based on clients, servers, DNS and central allocators[8]. With comparing the performance of various algorithms, the contribution provides the solid support to load balancing research. Referring to the commercial load balancing products, a lot of Chinese and international companies are making an effort to develop the software-based and hardware-based load balancing products, such as LVS, Lander Balance, Check Point, etc[9].

III. Dynamic Feedback-based Load Balancing Methodology

When facing a large number of concurrent service requests from users, the core task of implementing load balancing is to arrange the strategy through proper task scheduling, so that the server cluster can meet the requirements of the overall service duration, task throughput, resource utilization efficiency, scalability and many other constraints[10]. The dynamic feedback-based load balancing method proposed in this paper focuses on the load conditions and the dynamics of access tasks within each work node, and then it provides the task scheduling plan and the near-optimal execution solution under the concurrent request working conditions.

This paper elaborates the load balancing method with considering the following factors[11]:

(1) Node parameter: it refers to the real-time load status of each working node, including the number of tasks in the running queue, the speed of the system call, the size of the free memory, etc.

(2) Evaluation strategy: it refers to assess whether to re-transfer the follow-up tasks to other nodes for further processing according to the current workload of the working node. In this paper, we select to employ a threshold-based approach to determine the task transfer.

(3) Transfer location: given the tasks that are suitable for transferring to other work nodes, it is necessary to specify the target work node for task transfer.

(4) Maintenance means: it refers to determine the list of tasks to be transferred. Then the load scheduling plan for transferring tasks is executed maintain load balancing.

Through analyzing the above four factors, we can suggest the procedures for load balancing process. First, collecting and identifying the relevant available resources during the balancing execution[12]. The resource concerns the available working nodes, processing capabilities of nodes, available storage space and memory[13]. Second, analyzing and determining the execution condition of dealing with the arrived tasks. If all the tasks have been completed, then the node continues with waiting for new tasks. If there are some pending tasks, then the tasks are analyzed by concerning the arrival rate of tasks, the number of the tasks and the memory occupancy of the tasks[14]. Through considering the three factors, the strategy and the relevant parameters of dealing with tasks are adjusted. Third, collecting the parameters of tasks arriving at the working nodes to determine the current processing performance. In this way, the node and the feasible moment to start the load balancing process are settled[15]. Then configuring and executing the load balancing method, i.e., selecting the task items to be moved out, the working node where the items are located and the destination nodes to move in the tasks. Fifth, determining whether there is a new task to be moved in. If there is a new task, then identifying and collecting the available resource as the procedures in the previous step. If there is no new incoming task, then completing the current load balancing tasks (Seen in Figure 1).

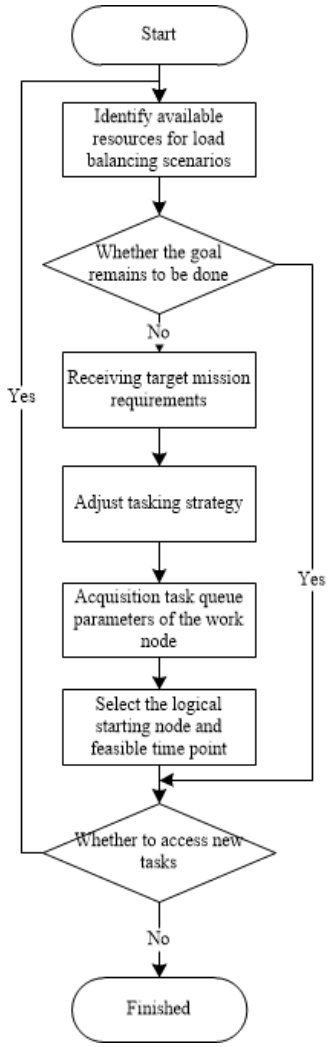

Fig.1. The execution flow of load balancing method

3.1 Dynamic feedback strategy

Given the actual situation of parallel processing, this paper proposes a global load balancing quality-oriented dynamic feedback strategy with considering the key factors under the load balancing scenario and the interrelations among the factors. During the process of accessing the service requests and making the response,

the load condition of all the nodes is collected and analyzed continuously. Combined with the dynamic feedback strategy, the balancing procedures are kept available and useful.

This paper proposes to dynamically analyze the current status of the working nodes relying on the feedback control theory for evaluating the load indicators to adjust and arrange the load balancing scheme, with which the load balancing solution is thus arranged and planned to gain the continuous effectiveness. (Seen Figure 2).

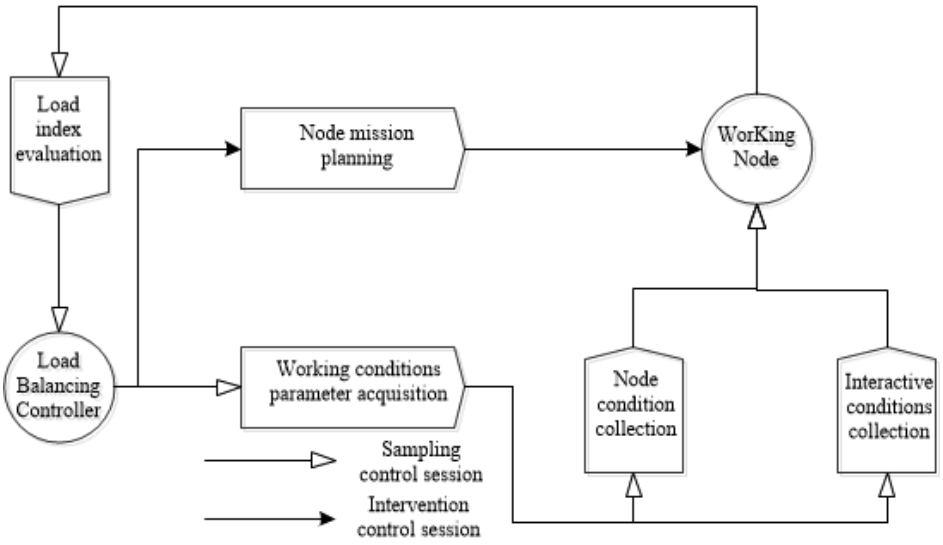

Fig.2. The evaluating scheme of the dynamic feed

In this paper, the working status of each node is taken as the starting point to continuously evaluate the resource occupancy and then to analyze various characteristic data during the task execution. Through continuously assessing the performance of the server cluster system, the task allocation plan is adjusted accordingly. Here are two specific tasks:

(1) to evaluate the load status of the work node;

(2) to formulate the load threshold setting strategy according to the load assessment and to dynamically adjust the threshold value correspondingly.

3.1.1 Evaluating load status of node

Node load assessment mainly concerns evaluating the working node's situation and the interactions between the nodes. The corresponding working parameters are:

(1) Working node operation indicator (denoted as S_N^i, *i* refers to the identity of the current work node, *N* represents the indicator is used to specify the parameter of a working node): This indicator reflects the performance of the current working node under the corresponding load condition and mainly covers the status of CPU (denoted as p), memory (denoted as r) and network card (denoted as t) that are participating in the task execution. This indicator reflects the performance of the current working node under the corresponding load condition and mainly covers the status of the core components participating in the task execution, such as CPU (denoted as p), memory (denoted as r) and network card (denoted as t). Meanwhile, the status (denoted as S_E^c) of the other component (denoted as c) that could affect the performance would also be concerned if necessary.

Because the instantaneous CPU load cannot truly reflect the real working conditions during the continuous execution of computing tasks, this paper selects to use the average CPU working condition assessment method based on moving inspection time window to calculate the CPU work when calculating CPU performance. With this method, the CPU usage conditions are collected at the start time point and end time point of the inspection time window. Then the overall CPU usage is calculated as the percentage of the CPU runtime during the inspection window out of the total CPU runtime. We take the percentage value as the assessing result of investigating the CPU (denoted as $usage_{cpu}$ and formulated as eq.(1)).

$$usage_{cpu} = (1-(idle_2 - idle_1)/(cpu_2 - cpu_1)) \times 100\% \quad (1)$$

In eq.(1), the variable denoted as idle represents the idle time of CPU at the corresponding time point, and the variable denoted as cpu represents the corresponding total CPU runtime.

In calculating the memory work condition (i.e., S_E^r), the percentage of the actual memory cost spent in the calculation out of the total memory capacity is used as the memory assessment result (denoted as $usage_{mem}$ calculated by eq.(2)).

$$usage_{mem} = (MemTotal - MemFree)/MemTotal \quad (2)$$

In eq.(2), MemTotal represents the total physical capacity of memory and MemFree represents the memory that is not used yet.

During calculating the network card work condition, we employ a network test tool to measure the data flow within the inspection time window. The network card work condition is assessed as a percentage of the maximum data flow value out of the network card bandwidth parameter.

Based on the working status assessments onto all the components within one working node, the working condition of the working node is expressed as:

$$\begin{cases} S_N^i = \sigma_p \cdot S_E^p + \sigma_r \cdot S_E^r + \sigma_t \cdot S_E^t + \sigma_c \cdot S_E^c \\ \sigma_p + \sigma_r + \sigma_t + \sigma_c = 1 \end{cases} \quad (3)$$

In eq.(3), σ represents the weight of the corresponding component during the measurement and it is called ***component performance weight***. All the working conditions of all the components are notated in form of percentage (from 0% to 100%). 0% indicates that the components are not used, whereas 100% means the component has reached the full load working condition.

(2) The node interaction indicator (denoted as S_I^i, *i* refers to the identity of the current work node, *I* represents the indicator is used to specify the interaction among the working nodes): the indicator reflects the communication performance between the current working node and the master node (i.e., the node in responsible for managing the distribution of tasks to all the other work nodes).

The interaction indicator is expressed as:

$$\begin{cases} s_I^k = \begin{cases} 0 & disconnected \\ 1 & connected \end{cases} \\ S_I^i = \dfrac{\sum_{k=1}^{m} s_I^k}{m} \end{cases} \quad (4)$$

In eq.(4), m represents the sum number of the links from the current node to the other ones; s_I^k represents the specific communication status and conditions from the current node (whose identity is k) to the other ones. If the interaction exists (i.e., the link is connected), then the value of s_I^k is assigned with 1. Otherwise, the value is assigned with 0.

With the above indicator, the load status assessment of each working node is expressed as:

$$S_L^i = \theta_N \cdot S_N^i + \theta_I \cdot S_I^i \quad (5)$$

In eq.(5), θ represents the weight of one indicator during the assessment and is called node performance weight.

3.1.2 The strategy of configuring load threshold

Through the computing process mentioned above, the strategy of scheduling tasks to the working nodes can be established with the above load indicators, during which we employ two thresholds as the scheduling decision parameters, i.e., δ_1 and δ_2 ($\delta_1 < \delta_2$).

(1) When $S_L^i < \delta_1$, it means that the working load of the current working node is relatively small. The working node can complete the scheduled even earlier and it could be assigned with new tasks in advance.

(2) When $\delta_1 < S_L^i < \delta_2$, it means that the working load of the current working node is normal. The working node can finish the tasks as planned and it could be assigned with the new tasks requiring small load.

(3) When $S_L^i > \delta_2$, it means that the working load of the current working node is too large. And the working node should not be assigned with new tasks in order to prevent the overload that might lead to server crash. In this case, after completing the current task, the load assessment indicator would be reset as 0. The node should re-participate in the load information feedback process to obtain the new tasks in the further scheduling.

Through analyzing eq.(3), eq.(4) and eq.(5), the load indicator denoted as S reflects the load status and working conditions of the nodes as well as their interactions. The component performance weight denoted as σ and the node performance weight denoted as θ reflects the criticality of either load condition (i.e., node working condition or component working condition) during feeding back the assessment.

Therefore, the configuration of the weight parameters can introduce the load balancing scenario into the process of formulating the schedule scheme. The weight value mains origins relying on the experiences from the data center engineers and they would be gradually turning to stable through long-term running. Moreover, the paper recommends to initialize the two scheduling decision parameters (i.e., δ_1 and δ_2) as 0.5 and 0.8 respectively based on the authors' practices.

During load balancing process, the load parameters are continuously collected to form the periodic assessment onto the load indicators. Meanwhile, the period of collecting parameters can also be adjusted corresponding to the scenarios where to run load balancing solution. The adjustment can prevent the larger consumption of the system performance and the energy. According to the practical experience, we recommend to configure the collecting period between 10 seconds and 90 seconds corresponding to the service request frequency.

During operating the dynamic feedback-based load balancing method, we find that the situations that lead to the dynamic adjustment of tasks in the cluster and invoke the feedback actions mainly occur in three scenarios, i.e., new tasks are loaded in real-time, real-time dynamic changes of network bandwidth is dynamically changed in real-time, and instant load capacity of working nodes is being adjusted.

During loading the new task, the joining of newly added working node(s) will not have a negative impact on overall mission planning and cluster performance if the new working nodes are included in the load balancing solution. The newly added work nodes will be given the priority to perform tasks with a relatively short time and are configured to complete and implement the corresponding feedback measures according to the above strategies.

3.2 Task scheduling in load balancing

In the server cluster that implements the load balancing solution, the load balancing algorithm deployed at the master node considers the real-time load conditions and processing performance of all working nodes, and it constantly adjusts the proportion of task distribution to avoid various problems caused by unbalanced task distribution . Because the load balance dynamically changes with time, the workload of each working node is roughly equal to the load threshold through the dynamic adjustment of the threshold, so that the tasks can be evenly distributed and the working efficiency of each working node can be brought into full performance.

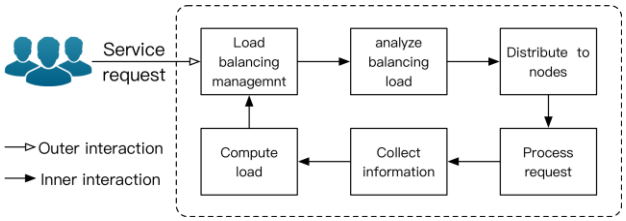

Fig.3. Working process of load balancing

The load balancing process includes (Seen in Fig.3):

(1) Load balancing management is carried out on the master control node, and the load of the system is analyzed according to the working status of the existing nodes and the load balancing threshold.

(2) Through the connection between the nodes, the master control node distributes the work tasks to each working node.

(3) Each working node processes the received tasks.

(4) The real-time load status information for each worker node is collected.

(5) The working condition of each node is calculated and evaluated according to the collected load status information.

(6) Through comparing the working status of the node with the load balancing threshold, the load balancing threshold is adjusted according to the preset balancing strategy in the further load balancing management.

During the information collection phase of load balancing, the master control node collects information about the real-time operation of other work nodes. The collected information mainly includes the load of the node, the task allocation, and the response speed. The collection phase is executed periodically according to the strategy and the actual situation. The more commonly used mechanism is that the working node periodically sends information to the master control node. The working node periodically sends state information to the master control node by using the heartbeat mechanism to ensure that the master control node has a newer working node status and enables the former to use this Judge whether the working node exists.

Periodic information collection is relatively simple to implement, but the problems are relatively obvious. First, the periodic information transmission will increase the load of the master control node. Since all the working nodes send messages to the master control node, it will cause greater communication overhead. Second, the length of the message transmission is not easy to determine. Too short period will cause the increase of communication load whereas too long period will cause the update is not executed in time. Third, if we send the new tasks within a task allocation cycle for many times, then it will cause the further tasks cannot be efficiently processed. Conversely, when there are no new tasks arriving between the two updating actions, the system resources are wasted.

In view of the above situation, the periodic information collection of the master node is not entirely suitable for various situations occurring in practical applications. Therefore, this paper extends the method of collecting periodic information so that the master control node collects the status information of the working nodes according to the needs of the current task data along collecting the periodic information. The idea of collecting information according to the number of tasks is that when the master control node receives a new task, it actively sends a status query request to the working node and collects the workload of the working node. When the working node receives the status query request from the master control node, it will send the current load information, resource occupation information and task allocation information to the master control node.

The scheduling process is stated as follows:

```
if (new task arrives)||(a new period starts)
        reset clock;
        the master node sends out query request and receives the load information from the working nodes
        if (the current node denoted as i satisfies the scheduling constraints)
                assign task to node i or add the node into queue
        end if
end if
```

The process of the current working node (denoted as i)
if the node receives the query request from the master control node
compose the load information to the master control node
end if

In the interaction process among all the nodes, TCP protocol can be used for message transmission to ensure reliability. According to the transmission characteristics, the working node accepts the task assignment from the master control node and replies to the master control node. In order to ensure the correctness of the information replying to the master control node, After receiving the task, the working node replies a message to the master control node, which contains the working node's own information and the partner node information used in the load balancing adjustment process, and each working node adds the carried time in the reply message. The message is insert with the timestamp to ensure that the information received by the master node is up to date. The master control node updates the stored corresponding working node information according to the carried information, which can not only catch the working node load information accurately but also reduce the communication load between the nodes and keep the scheduling algorithm simple and feasible. When the system is idle, the working node waits for the master control node to collect information and make decisions. When the system is running, the working nodes process the actual tasks and replies in parallel. The working nodes do not continue to wait for the allocation of tasks in order to take full advantage of system resources.

The master control node collects information only when it submits a task. When the working pressure of the system load balancing is relatively high and the number of tasks performed during a specific time period is too large, the master control node consumes more resources and more time in task allocation. According to the algorithm, the working node would send the latest its own load information to the master control node after receiving the task, the latter only updates the information of the nodes that have accept the tasks. Referring to the working nodes that haven't received the tasks, they would not send the updated information to the master control node. In this way, it is possible to effectively reduce the possibility of idle nodes receiving new tasks. In combination with the periodic collection of node load information, the master control node periodically collects the information of all the nodes while maintaining the on-demand collection manner, and increases the possibility of accepting new tasks for nodes that have not been assigned tasks in the near future. In summary, the combination of periodic collection and on-demand collection of work patterns, to ensure real-time load information and reduce the amount of interaction between nodes to improve the overall performance of the system at the same time, better balanced load of each node.

IV. Verification Experiments

In this paper, we adopt a cloud computing simulation platform— CloudSim to verify load balancing solutions (version 4.0). Based on the discrete event model, CloudSim is an application system for simulation, which is developed by Java language. Thus, it has unique features of Java language such as cross-platform deployment. It can run on different operating systems like Windows, Linux or MacOS. In addition, it can be used for modeling and simulating for system architectures and deployment plans in a cluster environment.

In this work, we utilize some core components of CloudSim to build the simulation environment, which includes the following steps:

(1) DataCenter class: used for simulating the solutions of data center located in the cluster infrastructure, and encapsulating the related methods to configure the corresponding work nodes.

(2) DataCenterBroker class: used for encapsulating the related methods to manage the inside work nodes.

(3)Host class: used for simulating the mapping relations between the physical hosts and the virtual hosts in cluster environment.

(4) VirtualMachie class: used for simulating the deployed virtual hosts in cluster. The virtual host acts as a member in Host class to simulate the resource sharing and internal scheduling.

(5) VMScheduler class: used for simulating the policies of dispatching and managing among various virtual machines.

(6)VMProvsioner class: used for configuring the mapping relationship between Host object and VirtualMachine object belonging to DataCenter objects.

(7) Cloudlet calss: used for simulating the tasks in cluster, and configuring the resources.

In this work, we took advantage of three physical hosts (identified by PH_LB_1601 、 PH_LB_1602 and PH_LB_1603) to configure the load balancing simulation environment. Each physical host installed and configured JDK8.0, configured CloudSim4.0, and set environment variables.

The load balancing simulation environment set the three physical hosts mentioned above to one DataCenter object (identified by CS_DC_LB). The host PH_LB_1601 set two VirtualMachine objects (identified by VM_CL_1601_01 and VM_LB_CTL_1601_02 separately) in DataCenter object as a load balancing controller, used for dispatching and managing the load balance. Physical hosts PH_LB_1602 and PH_LB_1603 build three VirtualMachine objects separately (identified by VM_LB_1602_01 、 VM_LB_1602_02 、 VM_LB_1602_03 、 VM_LB_1603_01 、 VM_LB_1603_02 and VM_LB_1603_03). The inside VirtualMachine object creates a load balancing solution consisting of seven nodes. The network topology of the above nodes is illustrated as Figure 4:

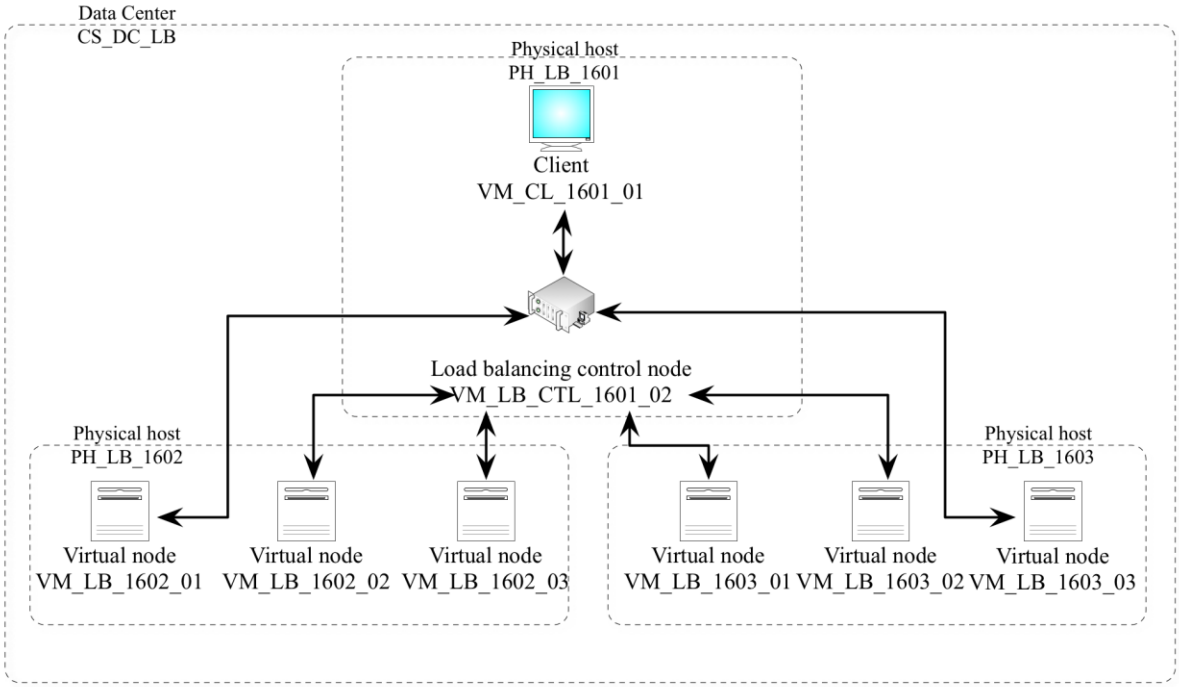

Fig.4. The network topology of the experiment

Based on the afore mentioned network topology, we called the Host object and VMScheduler object to deploy the load balancing strategy and scheduling algorithm based on Dynamic Feedback to the physical hosts PH_LB_1601, PH_LB_1602 and PH_LB_1603.

For testing the load balancing scheme, this paper configures the load task generated by Httperf in the VirtualMachine object VM_CL_1601_01 in order to test the accessing capability of service request. The test scenario focuses on two performance measures: (1) the association between the average response time of service requests and the number of concurrent connections to reflect the service access performance of the solution; (2) the number of available concurrent connections in a given service request scenario to reflect the maximum concurrent access capability. In the experimental environment, the average response time of a cluster consisting of multiple servers to multiple concurrent connections is relatively stable, and the test data is shown in Table-1. From the above experimental results, it can be seen that the cluster-based load balancing scheme has good performance in handling concurrent requests. When the number of concurrent connections continues to rise, the response time remains within 7 ms. It can be seen that the load balancing scheme built by clustering can effectively control the server to process the request access time.

Table-1. The response of load balancing solution

Concurrent connections	200	400	600	800
Response time（ms）	3.4	3.5	3.9	6.7

For evaluating the load balancing strategy, we mainly studied the actual number of concurrent connections to analyze the performance of the load balancing algorithm based on dynamic feedback, and compared it with the Nginx-based IP Hash algorithm. Among them, the core idea of IP Hash algorithm is to hash map according to the IP address of the service request source, and to use the result of hash operation as the basis to select the server node that actually answers the service request, and then to allocate the service request to the corresponding server node (See Table-2).

Table-2. Actual concurrent connections between the algorithms

Concurrent connections	IP Hash algorithm Actual concurrent connection amount (access rate)	Dynamic feedback-based load balancing algorithm Actual concurrent connection amount (access rate)
200	200（100%）	200（100%）
400	399.8（99.9%）	399.8（99.9%）
600	599.2（99.9%）	599.1（99.9%）
800	798.2（99.8%）	794.9（99.4%）
1000	682（68.2%）	980.4（98.0%）
1200	562.1（46.8%）	734.2（61.2%）
1400	513.7（36.7%）	662.2（47.3%）

Figure 5 shows that when the number of concurrent connections is low (lower than 800), the actual number of concurrent connections is basically the same as the number of concurrent requests. With occasional small loss (the rate of loss of IP Hash algorithm is 0.2% and the loss of load balancing algorithm based on the dynamic feedback is 0.6%), it can maintain the basic normal request access performance. When the number of concurrent connections continues to increase and reaches 1000, the IP hash algorithm shows a significant loss of concurrent connection requests, which results in a loss

rate of 31.8%. In contrast, the rate of loss based on dynamic feedback load balancing algorithm is only 2%. In conclude, the latter has better service request access performance.

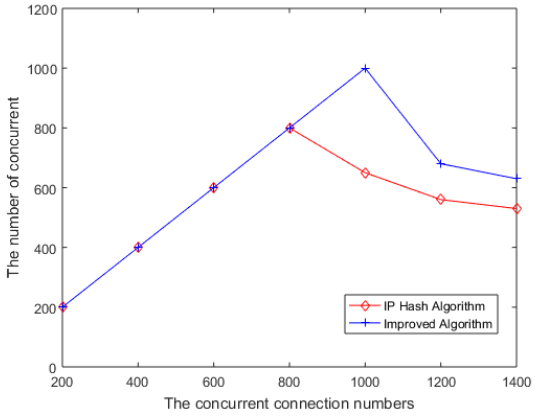

Fig.5. Comparison of actual concurrent connection amount

V. Conclusion

In this paper, the dynamic feedback-based load balancing method is introduced to optimize the dynamic feedback strategy in the process of dynamic load balancing. In the real case, load balancing task scheduling is optimized and task queue scheduling in load balancing is analyzed in detail. The paper first realizes an assessment of the load of the nodes and then optimizes the load balancing optimization strategy by setting the threshold of the load balancing. The experimental results are verified by setting up an experimental environment. It is proved that the optimized load balancing algorithm based on dynamic feedback is better than IP Hash algorithm in solving high data concurrency. With the increasing amount of data in the network, the research on network load balancing optimization requires more optimized algorithms in the process of data high concurrent processing. More research will be conducted in this respect in future work.

Acknowledge

This work is supported by Jilin Provincial Science and Technology Department via International Joint Research Project (20150414055GH).

Reference

[1] Al-Anbagi I, Erol-Kantarci M, Mouftah H T. A low latency data transmission scheme for smart grid condition monitoring applications[C]// Electrical Power and Energy Conference. IEEE, 2013:20-25.

[2] Zhong C, Cai D, Yang F. Divisible loads scheduling using concurrent sending data on multi-core cluster[J]. Journal of Computer Research & Development, 2014, 51(6):1281-1294.

[3] Wang S C, Yan K Q, Liao W P, et al. Towards a Load Balancing in a three-level cloud computing network[C]// IEEE International Conference on Computer Science and Information Technology. IEEE, 2010:108-113.

[4] Bhogal K S, Lewis P D, Okunseinde F O, et al. Computer data communications in a high speed, low latency data communications environment[J]. 2015.

[5] Scharber J. CLIENT-SELECTABLE ROUTING USING DNS REQUESTS:, US20150264009[P]. 2015.

[6] Stoltzfus A, O'Meara B, Whitacre J, et al. Sharing and re-use of phylogenetic trees (and associated data) to facilitate synthesis[J]. Bmc Research Notes, 2012, 5(1):1-15.

[7] Leber A W, Knez A, Von Z F, et al. Quantification of obstructive and nonobstructive coronary lesions by 64-slice computed tomography: a comparative study with quantitative coronary angiography and intravascular ultrasound.[J]. Journal of the American College of Cardiology, 2005, 46(1):147.

[8] Cardellini V, Colajanni M, Yu P S. Dynamic Load Balancing on Web-Server Systems[J]. Internet Computing IEEE, 1999, 3(3):28-39.

[9] Andrews J, Singh S, Ye Q, et al. An Overview of Load Balancing in HetNets: Old Myths and Open Problems[J]. IEEE Wireless Communications, 2013, 21(2):18-25.

[10] Peng X. Research and Realization of Task Scheduling in Coupling Distributed System[J]. Computer Technology & Development, 2013.

[11] Bao-Tong D U, Bing L I, Yang R. Concurrent service composition approach based on QoS fuzzy dominance[J]. Computer Engineering & Design, 2015.

[12] Debankur M, Borst S C, Van L J S H, et al. Universality of load balancing schemes on the diffusion scale[J]. Journal of Applied Probability, 2016, 53(4):1111-1124.

[13] Kai H, Shen H. Locality-Preserving Clustering and Discovery of Resources in Wide-Area Distributed Computational Grids[J]. IEEE Transactions on Computers, 2012, 61(4):458-473.

[14] Zhang N, Yan Y, Xu S, et al. A distributed data storage and processing framework for next-generation residential distribution systems[J]. Electric Power Systems Research, 2014, 116(11):174-181.

[15] Yang W C, Huang W T. A load transfer scheme of radial distribution feeders considering distributed generation[C]// Cybernetics and Intelligent Systems. IEEE, 2010:243-248.

Authors' Profiles

Xin ZHANG, Ph.D., lecturer at School of Computer Science and Technology in Changchun University of Science and Technology. His interests include: Mobile Internet, design engineering, software engineering and system engineering.

Jinli LI, Master candidate at School of Computer Science and Technology in Changchun University of Science and Technology. Her research interests include Internet of Things and applications, computer network, software engineering.

Xin FENG, Ph.D., Associate Professor of School of Computer Science and Technology in Changchun University of Science and Technology. His research interests include Internet of Things technology and applications, software engineering and information system, database and data mining.

How to cite this paper: Xin ZHANG, Jinli LI, Xin FENG, "A Dynamic Feedback-based Load Balancing Methodology", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.12, pp. 57-65, 2017.DOI: 10.5815/ijmecs.2017.12.07

