Adaptive Load Balancing Algorithm Using Service Queue

Gowtham Kanagaraj, Naveen Shanmugasundaram And Sathish Prakash.

Abstract— In this internet driven age it is difficult to meet the demands of the people in E commerce. This is fixed by using the load balancing techniques. Load balancing refers to distributing incoming HTTP requests across web servers in a server farm. This is achieved by various load balancing algorithms like Round robin, Weighted Round robin, Observed, Fastest etc. Even though objective of load balancing is achieved in recent years; still placing of computational logic and decision making and zeroing in for the best server has not attained its efficiency. It is under study. We suggest a method to dynamically load balance using service queue wherein every server computes its load value by summing the load parameters like memory utilization, CPU utilization, and network utilization and exchange load value with central node in a certain cyclic period. Central node selects the least loaded server among farm of servers to process the request. Each server at central node waits in a queue called service queue for its turn to process the request.

Keywords— static load balancing, dynamic load balancing, central node, load value, load parameters, memory utilization, CPU utilization and network utilization.

I. INTRODUCTION: LOAD BALANCING – AN OVERVIEW:

WITH the rapid development of the Internet and the increase in the number of netizens, one server node may be accessed by a lot of users simultaneously. This situation has result in heavy demands on the sever node, because the single server cannot control its load from user requests in time. So the cluster computer has been already proposed. The control of a cluster system can be centralized and distributed. The centralized clusters are discussed in the paper because it is simple and can save the price of the communications between the nodes [1] [2]. The key question of a cluster computer is the load balancing. Load balancing is responsible for allocating user requests to the server nodes in order to prevent the emergence of the situation that some server nodes are loaded heavily while others are loaded lightly.

Load balancing can provide an efficient and safe method for the server nodes to process the user requests. The main goals of the load balancing are to minimize the total response time and maximize throughput. And all operations are transparent to users. Load balancing algorithms generally can be classified as either static or dynamic [3] [4] [5]. Static algorithms only use given node information to make load balancing decisions. They cannot be true and comprehensive for the real load variation of the server nodes. Although dynamic algorithms are able to dynamically reflect the load of the server nodes, it is very difficult to implement. Static algorithms include the round robin algorithm, the weighted round robin algorithm and so on. And dynamic algorithms include the least connections algorithm, the improved multi-parameter load balancing algorithm.

II. RELATED WORKS

Load Balancing Algorithms:

Load balancing algorithm directly influences the effect of balancing the server workloads. Its main task is to decide how to choose the next server and transfer a new connection request to it. We know that load balancing algorithm is divided into static algorithm and dynamic algorithm. The static algorithm is easily carried into execution and takes less time, which doesn't refer to the states of the servers [6]. The common static algorithms are Random Algorithm, Round-Robin Scheduling Algorithm, and Weighted Round-Robin Scheduling Algorithm etc. In a random Scheduling, the requests are assigned to any server picked randomly among the group of servers. In such a case, one of the servers may be assigned many more requests to process, while the other servers are sitting idle. However, on average, each server gets its share of the load due to the random selection. It is simple to implement, but it can lead to overloading of one server while under-utilization of others.

Round-Robin Scheduling Algorithm is that the IP sprayer assigns the requests to a list of the servers on a rotating basis [7]. The first request is allocated to a server picked randomly from the group, so that if more than one IP sprayer is involved, not all the first requests go to the same server. For the subsequent requests, the IP sprayer follows the circular order to redirect the request. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned. The Round-Robin Scheduling Algorithm is better than random allocation because the requests are equally divided among the available servers in an orderly fashion; however, it is not enough for load balancing
based on processing overhead required and if the server specifications are not identical to each other in the server group. Weighted Round-Robin Scheduling Algorithm is an advanced version of the round-robin that eliminates the deficiencies of the plain round-robin algorithm [7].

In case of a weighted round-robin, one can assign a weight to each server in the group so that if one server is capable of handling twice as much load as the other, the powerful server gets a weight of 2. In such cases, the IP sprayer will assign two requests to the powerful server for each request assigned to the weaker one. The Weighted Round-Robin Scheduling Algorithm takes into account the capacity of the servers in the group but it does not consider the advanced load balancing requirements such as processing times for each individual request and its time complexity is higher than the Round-Robin Scheduling Algorithm. The dynamic algorithm is self-adaptive algorithm, which is better than static algorithm [8]-[11].

Self-adaptive load balancing system mainly includes two processes: monitoring the load states of servers and assigning the request to the servers [12]. The state supervision, which depends on the load information of each server in the cluster monitored and collected periodically by the front-end balancer, raises the effect of load balance by monitoring load variety, however, this will burden the workload of balancer which is the bottleneck of the cluster system. The dynamic algorithm often bases on the static algorithm, so the paper mainly discusses the latter. According to the analysis above, an ideal load balancing algorithm should achieve the following targets: Reduces the time complexity of load balancing algorithm as far as possible, also reduce the workload of the front-end balancer and get a better load balancing effect.

III. ALGORITHM DESCRIPTION

A. Introduction to Adaptive Load Balancing Algorithm using Service Queue:

Adaptive load balancing algorithm attempts to balance the workload dynamically by responding to the current system state. It should be able to improve cluster performance. However these algorithms are usually more complex than static algorithms. They collect information on the current cluster state and make decisions on process allocation based on that information. The architecture of the algorithm is illustrated in fig 1.

When client makes a request, it hits the central node which decides the server to allocate for processing the server based on the data available in the load table. We can characterize the responsibility of the client, central node and server as follows,

\[
\text{Role of client:} \quad \text{Request the server for a page or file or process.}
\]

\[
\text{Role of central node:} \quad \text{It maintains the load table which has the information about the load value of servers and its load parameters. Central node receives the client request looks up for the least loaded server in load table and assigns the respective server to process the request.}
\]

\[
\text{Role of server:} \quad \text{Its primary role is to process the request. Apart from it also maintains a load parameter table which has the information of the local machine i.e. server. It updates the data in certain cycle and exchange to the central node. If the load value of server is high then it doesn’t update since it’s overloaded. So it waits till it comes back to normal.}
\]

IV. ALGORITHM DESIGN

The Adaptive Load Balancing algorithm basically does the following,

\[
\text{Load} = \begin{cases} 
\text{idle} = \text{cpu_u < 1}\% \\
\text{high} = (\text{mem_u} > 85%) \text{or (cpu_u is high)} \\
\text{low} = (\text{cpu_u is low and n_u is low}) \text{or} (\text{n_u is low}) \\
\text{normal} = \text{otherwise} 
\end{cases}
\]

\[
\text{(4)}
\]

\[
\text{Classify node.}
\]
• Selection process.
• Process the request.

**Load estimation and exchange:**
Ideally, the load information should reflect the current CPU utilization, memory utilization and network traffic of a node. These are the metrics on which load value is decided. Consider the metric CPU utilization; it may be high in one second but low in the next second. Therefore it is reasonable to average these statistics over several seconds. Similarly each metric is calculated. Thus load value can be expressed mathematically in equation (1)

\[
\text{Load}_n = \sum (\text{cpu}_u, \text{mem}_u, n_u)
\]  

(1)

Where,
- \(\text{Load}_n\) – Load value of the node \(n\).
- \(\text{cpu}_u\) – Average of CPU utilization over a period of 5 seconds.
- \(\text{mem}_u\) – Average of memory utilization over a period of 5 seconds.
- \(n_u\) – Average network traffic over a period of 5 seconds.

**Classify nodes:**
Classification of node based on the load values involves taking average of each load metric of each node. \(\text{Load}_u\), \(\text{Load}_m\), \(\text{Load}_n\) will have the average value of each node’s CPU utilization, memory utilization and network traffic respectively. These values are used to calculate the threshold values. Threshold values can be expressed in equations (2) & (3)

\[
\begin{align*}
\text{tH} &= H \times \text{Load(par)}_{\text{avg}} \\
\text{tL} &= L \times \text{Load(par)}_{\text{avg}}
\end{align*}
\]  

(2) & (3)

Where,
- \(\text{tH}\) - high threshold
- \(\text{tL}\) - low threshold
- \(\text{Load(par)}_{\text{avg}}\) - Average value of each node’s Load parameter where par can be \(\text{cpu}_u\), \(\text{mem}_u\) and \(n_u\)
- \(H\) - Constant greater than 1
- \(L\) – Contanst less than 0.

**Selection Process:**
Using the threshold values of each parameter, the nodes will be grouped as idle, low, normal or high according to the following criteria. For each node, the CPU utilization, memory utilization and network traffic will be checked to decide whether it is in idle, high, low or normal level. The classification criteria is as follows in equation (4),

\[
\begin{align*}
\text{Idle:} & \\
\text{High:} & \\
\text{Low:} & \\
\text{Normal:} &
\end{align*}
\]

Running such as daemons or monitoring programs. Most of background programs run periodically for a very short period of time. We set the CPU utilization to 1% to ignore these background programs. When the CPU utilization of a node is less and equal than 1%, the node is considered as an idle node.

**High:**
If any of these three conditions is true, the node is classified in the high state.

**Low:**
If either of these two conditions is true, the node will be classified in the low state.

**Normal:**
Nodes that are not classified into one of the other categories are the nodes in the normal load state. They are considered as more loaded than the low state and less loaded than the high state. Based on the above selection process there may be a query if all server will get a possibility to serve the client request or in other words will all server be a best server in some time. In probability theory and statistics, the discrete uniform distribution is a discrete probability distribution that can be characterized by saying that all values of a finite set of possible values are equally probable. If a random variable has any of \(n\) possible values that are equally probable, then it has a discrete uniform distribution. The probability of any outcome \(Ai\) is \(1 / n\) [13]. Thus assume a scenario wherein three servers are to serve client request as per DUD (Discrete Uniform Distribution) every server has equal probability to serve. Probability of each server to process is \(1/3\) in this case, where total probability is 1.

**Process the request:**
Once the load is classified based on classification the best node is selected by the central node. Best server process the request.

V. **WORKFLOW**

**Work flow:**
We saw the functionalities of the components in the architecture. Let us see the workflow of the algorithm. We can represent the flow in a data flow diagram as in fig 2.

Client makes a request. This request is received by the central node. The central node as we know has a load table which has the load values of the servers in the server farm. Now the central node checks for the least loaded server in the table. After which the respective server is assigned to process and generate the request.
VI. Conclusion

Adaptive load balancing algorithm is a method for load balancing where the parameters used are based on the current state. Here centralized node takes the decision of allocating server based on the input from each server. Thus servers also take part in deciding to process request, as a result the overall performance is expected to be efficient and at its best. Future works can be in identifying a different set of load parameters which can effectively balance request with less of computational work.

Acknowledgment

We would like to thank Dr. A. Srinivasan, Prof & Head, Department of Information Technology, Mrs. V. Nandhini, Assistant Prof, Department of Information Technology, faculty staffs and friends for their support and guidance in bringing out this paper.

References