Load Balancing in Cloud-based Flow Security System

Gang HE\(^1\) and Yue-jian XIAO\(^2\)

\(^1\)Beijing Key Laboratory of Network System Architecture and Convergence, Beijing, China
\(^2\)Beijing University of Posts and Telecommunications, Beijing, China

Keywords: Cloud computing, Load balancing, Join-Idle-Queue algorithm, Randomized algorithm.

Abstract. In the context of load balancing [1], Lu et al. proposed a novel class of algorithms called Join-Idle-Queue (JIQ) for distributed load balancing in large systems, where a group of dispatchers distribute jobs to a cluster of parallel servers. Each dispatcher maintains a queue called Idle-Queue for idle servers. When a job arrives to a dispatcher at random, the dispatcher sends it to a server which in the head of the Idle-Queue, or to a random server if the Idle-Queue is empty. Meanwhile conversely, when a server has no jobs, it actively requests to be placed on an Idle-Queue. Although this algorithm was shown to be quite effective [2,3], it is defective in some respects. This paper presents several new improvements, which makes the algorithm more effective, especially in Cloud-based Flow Security System (CFS). These improvements focus primarily on these aspects, such as the definition of idle server, data concurrency while choosing an Idle-Queue, etc. The experimental results show that the improved load balancing algorithm can improve the utilization of resources in a certain extent.

Introduction

Cloud computing is a fast-growing area in computing research and industry today. It provides on-demand access to distributed resources. Balancing load is one of the biggest issue that cloud computing is facing today [4]. The load be distributed fairly among all the nodes. Load balancing also promotes optimal utilization of resources and increase throughput.

Cloud-based Flow Security System is a kind of architecture to ensure that the Internet application's environment is safe, meanwhile, using cloud services to provide convenient and effective applications to business users. The architecture of CFS refers to Figure 1. In this architecture, the main things related to load balancing are tunnel [5] management, server cluster management, traffic collection and traffic reduction.

The tunnel management maintains the tunnel between the CFS platform and the probe or the mobile terminal agent. Under the instructions of server cluster management, tunnel management can adjust tunnels dynamically according to load balancing algorithm.

Server cluster management maintains a list of platform servers, including the addition and deletion of the server, the adjustment of server parameters as well as the migration of corresponding workload. There are different types of servers in the server cluster. Dedicated server clusters access tunnels from probe and mobile terminal agent, then extract the original data packages from it. Next, Traffic collection module sends data traffic to traffic reduction module. Another dedicated server clusters access the data traffic from the traffic collection module, then process traffic reduction and extract session payload. In terms of load balancing, different load parameters are captured for different types of server clusters so that the load of the server is dynamically adjusted according to the load balancing algorithm.
Based on the CFS platform, this paper chooses improved-JIQ as the load balancing algorithm.

The Original Join-Idle-Queue Algorithm

The Join-Idle-Queue Algorithm Description
This algorithm consists of two layers of load balancing. Idle-Queue is a data structure which Connect the two layers balancing together. Figure 2 illustrates the overall system with an Idle-Queue. An Idle-Queue is a list of a subset of processors that have reported to be idle. All processors are accessible from each of the dispatchers.

Figure 1. The architecture of cloud-based flow security system.

Figure 2. The JIQ algorithm with distributed dispatcher, which is equipped with an Idle-Queue.
In the first layer load balancing system, at a job arrival, the dispatcher consults its Idle-Queue. If the Idle-Queue is non-empty, the dispatcher removes the first idle processor from the Idle-Queue and directs the job to this idle processor. If the Idle-Queue is empty, the dispatcher directs the job to a randomly chosen processor.

In the second layer load balancing system, when a processor becomes idle, it chooses one Idle-Queue based on a load balancing algorithm and joins it.

**The Flaws of JIQ Algorithm**

Firstly, when the server cluster is at a high load, the number of idle processor are fewer than Idle-Queue [6] on average. In this condition, JIQ algorithm changes to the poor strategy of assigning a job to a processor chosen uniformly at random. Secondly, it is an asynchronous calculation that an idle processor chooses an Idle-Queue and joins it. When there are a lot of idle processor trying to choose Idle-Queue and joins it, since each idle processor is inclined to choose a smaller length of the Idle-Queue, a large number of idle processors are often stacked in an Idle-Queue whose length is small before. Thirdly, in the first layer load balancing system, when Idle-Queue isn’t empty, job can be assigned to an optimal processor which in the Idle-Queue rather than the first one. Obviously, these make the server cluster load unbalanced.

**The Improved-Join-Idle-Queue Algorithm**

**Definition of Integrated Load Information**

According to the defect of JIQ algorithm, the paper provides some improvements. Firstly, using the processor's integrated load information instead of just determining whether the processor is idle. The calculation formula of the integrated load information refers to Eq. 1. L is the integrated load information of processor, P is the performance of CPU, D is the capacity of disk, B is the network bandwidth. $u_1$, $u_2$, $u_3$ are respectively the usage rate of CPU, disk and network bandwidth. $w_0$, $w_1$, $w_2$ are respectively the weights of CPU, disk and network bandwidth, which reflects the importance of different types of resources. Here, the value of weights is the same as in [7], $w_0$ is 40%, $w_1$ is 20% and $w_2$ is 40%.

$$L = w_0 \times P \times u_1 + w_1 \times D \times u_2 + w_2 \times B \times u_3.$$  \hspace{1cm} (1)

**Definition of Idle Processor**

Since using L to reflects the processor's load information, there is a more rational way to determine whether the processor is idle. Any time a processor is not on an Idle-Queue, and its L falls to a threshold T or below, it places itself on an Idle-Queue. Increasing the threshold T appears potentially beneficial under very high loads. But, according to [2], this approach may be unfriendly to low load situations. Therefore, a dynamic threshold T is a good idea. The threshold T is managed by Idle-Queue. Its flow chart refers to Figure 3.
Concretely, Idle-Queue maintains a variable $N$ whose initial value is zero. When a job arrives at this Idle-Queue, the value of $N$ plus 1 when the Idle-Queue is empty, otherwise, the value of $N$ is subtracted by 1. If the value of $N$ is equal to a preset $N_{max}$, it appears that the Idle-Queue is empty in most cases and the server cluster is at high load. So, The Idle-Queue will notify all processors to increase the threshold $T$. Similarly, when $N$ meet to a preset $N_{min}$, it will notify all processors to lowering the threshold $T$. Hence, the threshold $T$ automatically adjusts its value according to the load of server cluster, a high threshold $T$ corresponds to a high load of server cluster, and the threshold $T$ is smaller with the load falls.

**Idle Processor’s Priority in Idle-Queue**

Instead of assigning job to the first processor, now, it's just as easy to find the processor which is minimum load in Idle-Queue. Assigning job to a minimum load processor promotes the load balancing.

**Data Concurrency While Choosing an Idle-Queue**

Firstly, use the sum of idle processor’s integrated load information (SL) in the Idle-Queue as the basis of choice for idle processor to join. That is an idle processor chooses $d$ random Idle-Queue and joins the one with the smallest SL. Since that is an asynchronous calculation, as described above, unbalanced load conditions are easy to occur. Using synchronous computing to solving this problem is a bad way, because of lots of communication between processors and Idle-Queue. Assume that $SL_i$ is the $i_{th}$ Idle-Queue’s SL, and there are $R$ idle processors to be allocated. These $R$ idle processor’s total $L$ is $RL$. Consider from a load balancing point, after idle processors are assigned, $SL_i$ is as equal to each other as possible. Assume that $PL_i$ is the sum of idle processor’s integrated load information that joins in $i_{th}$ Idle-Queue.

$$SL_i + PL_i = E_0, \quad (i = 1, 2, ..., d)$$

$$\sum_i^d PL_i = RL.$$  \hspace{1cm} (2)

The value of $PL_i$ can be calculated by Eq. 2 and Eq. 3. For example, $PL_1 = (RL + SL_2 - SL_1)/2$ and $PL_2 = (RL + SL_1 - SL_2)/2$ with $d = 2$. Then, each processor is assigned to $i_{th}$ Idle-Queue at a probability of $P_i = PL_i/RL$, and it will chose one of them at randomly. Specifically, if $P_i < 0$, $P_i$ will be reset to 0. When there are $S$ jobs arrival at each period, and the load of server cluster is balancing, $R$ is approximately equal to $S$.

**Experiment**

In this paper, we simulate a finite system with 500 processors, 50 Idle-Queue and using a processor sharing service discipline. Every idle processor chooses 2 Idle-Queue and joins one of them and jobs.
arrive at Poisson distribution. We focus on the mean response time for the JIQ and Improved-JIQ algorithms in different load on system.

![Figure 4. Mean response time for the JIQ and Improved-JIQ algorithms.](image)

Figure 4 shows that Improved-JIQ algorithms is also suitable for high loads on system, and do not affect the performance when system is on low load.

**Summary**

In this paper, we have presented a architecture called Cloud-based Flow Security System. In this architecture, there are different servers that need to be managed. So it is reasonable that using integrated load information to define the resource. On the basis of integrated load information, we propose a dynamically threshold \( T \), which automatically adjusts its value according to the cluster load situation. When idle processors choose Idle-Queue, we also propose a randomized algorithm according to the mathematical expectations of the load. Last, the result of simulation supports our point of view largely. In the future work, we have planned to achieve the conversion between different type of servers dynamically in CFS, which can improve the overall system resource utilization. More meaningfully, we can control probe or the mobile terminal agent by tunnel management according to server cluster load, and implement load balancing in different LANs.

**References**


