文章编号：2095-2163(2024)02-0150-06 中图分类号：TP391 文献标志码：A

基于改进布谷鸟算法的自适应负载均衡算法

李 民
(浙江理工大学 计算机科学与技术学院，杭州 330018)

摘 要：针对负载均衡算法在高负载请求下服务器集群效率不高，负载不均衡及低负载请求下影响服务器效率的问题，文中基于 Nginx 负载均衡服务器，提出一种基于改进布谷鸟算法的自适应负载均衡算法。该算法通过熵权法确定 CPU、内存、磁盘 IO 性能和网络带宽等各项负载指标的权系数，衡量各项负载指标对负载评价影响的重要程度，并基于 Nginx 加权轮询算法和服务器实时负载情况，设计了高并发负载情况下的动态负载均衡算法。引入转化阈值计算，通过引入基于种群熵的改进布谷鸟算法计算 Nginx 静态加权轮询算法，转变为动态负载均衡算法的转化阈值。通过实验测试，相较于 WRR 算法、least-con 算法和动态权重算法，该算法在响应时间和实际并发数等方面具有表现更好，在数值上相比于动态权重算法有 18%左右的提升，验证了本文算法具有更好的负载均衡效果。

关键词：负载均衡；Nginx；服务器集群；改进布谷鸟算法；动态算法

Adaptive load balancing algorithm based on improved cuckoo search algorithm

LI Min
(School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou 330018, China)

Abstract: In response to the inefficiency of current load balancing algorithms for server clusters under high-concurrency requests as well as the problems of load imbalance and low-concurrency requests, this paper proposes an adaptive load balancing algorithm based on an improved cuckoo search algorithm for Nginx load balancing servers. The algorithm uses the entropy weight method to determine the weight coefficients of various load indicators such as CPU performance, memory performance, disk IO performance, and network bandwidth, to measure the importance of various load indicators on load evaluation. Based on Nginx weighted round-robin algorithm and real-time server load conditions, a dynamic load balancing algorithm is designed for high-concurrency load situations. By introducing a transformation threshold calculation, the improved cuckoo search algorithm for information entropy is used to calculate the transformation threshold from Nginx static weighted round-robin algorithm to dynamic load balancing algorithm. The experiments illustrate that compared with the WRR algorithm, the least-conn algorithm, and the dynamic weight algorithm, the proposed algorithm performs better in terms of response time and actual concurrency with an improvement of around 18% compared to the dynamic weight algorithm. This verifies that the proposed algorithm has better load balancing effectiveness.

Key words: load balancing; Nginx; server cluster; improved cuckoo search algorithm; dynamic algorithm.

0 引 言

随着互联快速发展，用户增长数量爆发式增长趋势，对服务器提出了更高的要求。为提供服务器高性能和高可用性，服务器集群成为了高效和廉价的首选方案。在讨论服务器集群时，负载均衡问题成为了研究焦点。负载均衡技术通过均衡后端服务器的并发请求，将请求分发到多个服务器上，高效利用所有后端服务器，提高服务器集群性能[1]。Nginx 是高性能的反向代理服务器，内置很多传统负载均衡算法，如轮询、加权轮询、Ip_Hash、最小连接数等算法[2]。Nginx 自带内置负载均衡算法，在低并发请求下有着较好的作用。但在高并发请求下，由于服务器自身性能差异或者不同请求对资源消耗的不同[3]，静态负载均衡算法没有考虑到服务器的实时负载情况，可能造成资源分配不均等情况，使得服务

基金项目：浙江省重点研发计划项目(2020C03094)；浙江省教育厅一般科研项目(Y202147659)；浙江省教育厅项目(Y202250706)；浙江省基础公益研究计划项目(QY19E050003)。

作者简介：李 民(1997-)，男，硕士研究生，主要研究方向：计算机网络与分布式处理。

收稿日期：2023-03-02

器集群性能下降[4]。

现有的负载均衡算法大致分为静态负载均衡算法和动态负载均衡算法两类[5]。静态负载均衡算法主要作用于服务器集群在简单的工作场景下，发挥一定的作用，但在愈发复杂的网络环境下（高并发、高负债），凸显许多问题。如：点到点式的静态算法在实际的网络环境中往往会产生大量错误的峰值点，导致最佳结果判定出错，容易造成负载不均衡[6]。近年来，在研究者们的努力下，中国在动态负载均衡研究方面亦取得了不错的进展。谭畅等[7]提出了一种云中心基于 Nginx 的动态权重负载均衡算法，该算法基于加权轮询策略进行改进，考虑服务器本身硬件性能和工作负载情况设计权重，但该算法仅在高并发状态，无法体现低负载情况下算法的效率。李慧斌[8]通过基于预测阈值提出一种动态权值负载均衡算法，计算集群负载平衡度，以避免频繁修改权重造成服务器抖动，但算法每次都要预测阈值，当并发量较小时，预测算法反而增加了响应时间。胡逸飞等[9]提出通过改进遗传算法的负载均衡算法，改进遗传算法计算动静态负载均衡阈值，实现动静态算法的转化，但由于遗传算法的缺陷，可能导致陷入局部最优解的情况，影响算法整体的准确性[10]，在高并发环境下影响效率。

以上文献中的算法在实验环境的高负载情况下大都有一定的提升效果，但大多数算法均未考虑到在低负载情况下算法的效率高低问题，同时存在容易陷入局部负载计算最优解，导致服务器集群分配不均、负载不均衡等问题。为了解决当前负载均衡算法所存在的问题，本文提出了一种基于改进布谷鸟算法的自适应负载均衡算法。

1　算法分析与设计

1.1　服务器节点负载评价函数

不同的评价指标对服务器的负载均衡判断有着巨大的影响，评价函数可以更加合理的评估服务器集群中每个节点的负载信息。本文的评价函数采用加权和法，通过服务器节点的各个负载指标以及对应的权重系数所构成的评价函数。根据文献和实验，使用各个服务器节点的 CPU 使用率、内存、磁盘 IO 和网络带宽作为集群负载评价指标[11]。由负载评价指标与不同指标的权重系数得出服务器负载评价指标函数，式(1)：

$$f(x_i) = \sum_{j=1}^{n} w_j x_{ij} \tag{1}$$

其中，X_{ij}为各负载指标；i 表示集群中的第 i 个节点；j 表示节点中的第 i 个指标；w_j 是各个负载指标中的权重系数，$j = 1,2,\cdots,n$。

本文使用 L_{cpu} 表示 CPU 使用率，L_{m} 表示内存使用率，L_{IO} 表示内存使用率，L_N 表示网络带宽使用率。使用 k_{cpu}、k_{m}、k_{io}、k_{n} 分别表示 CPU、内存、磁盘 IO 和网络带宽权重系数。满足：$k_{\text{cpu}} + k_{\text{m}} + k_{\text{io}} + k_{\text{n}} = 1$，对于每个节点 $S_i \in \{S_1, S_2, S_3 \cdots S_n\}$ $(n > 1)$，服务器 $SL(S_i) \in \{SL(S_1), SL(S_2), SL(S_3) \cdots SL(S_i)\}$ $(n > 1)$，节点负载评价函数为式(2)：

$$SL(S_i) = k_{\text{cpu}} L_{\text{cpu}}(S_i) + k_{\text{m}} L_{\text{m}}(S_i) + k_{\text{io}} L_{\text{io}}(S_i) + k_{\text{n}} L_{\text{n}}(S_i) \tag{2}$$

权重系数的取值通过熵权法确定，熵权法在构建评价指标中的因子权重环节，作为客观评价赋权法得到了广泛的应用。通过计算熵值来判断负载指标的离散程度，衡量负载指标对负载综合评价的影响[12]。负载指标的离散程度越大，对应着该指标的权重就越大。熵权法确定负载权重指标的步骤如下：

(1)计算第 j 项负载评价指标下，第 i 台服务器值占该指标的比重，式(3)：

$$P_{ij} = \frac{x_{ij}}{\sum_{i=1}^{m} X_{ij}} \tag{3}$$

其中，X_{ij} 为正向指标或负向指标。对于正向指标而言，数值越高则评价结果越好；负向指标则相反。对 m 进行数据的归一化处理，P_{ij} 根据数值数据标准得出。

(2)计算第 j 项熵值，式(4)：

$$e_j = -k \sum_{i=1}^{m} p_{ij} \ln(p_{ij}) \tag{4}$$

其中，$k = 1/\ln(m) > 0$，$j = 1,2,\cdots,n$；满足 $e_j \geqslant 0$；

(3)计算各项负载指标的权重系数，式(5)：

$$\varphi_j = \frac{h_j}{\sum_{j=1}^{n} h_j} \tag{5}$$

其中，h_j 为信息熵冗余度，值为 $h_j = 1 - e_j$，$j = 1,2,\cdots,n$。通过求解上述权重模型，得到各个负载评价指标的权重系数，计算出服务器负载指标的权重系数 k_{cpu}、k_{m}、k_{io}、k_{n}。

1.2　动态负载均衡算法设计

本文基于服务器节点负载评价函数(式(2))，设计出一种在高负载情况下的动态自适应负载均衡

算法,其算法步骤描述如下:

(1)在任务请求到达前,系统根据集群服务器节点的 CPU、内存等性能指标,计算每个服务器节点的处理任务能力,并设置一个初始权值大小。权值越大表示节点性能越强,反之权值越小则表示节点性能较低。权值决定着服务器集群应将任务分配到哪个节点,实现负载均衡的关键在于调整节点的权值。

(2)对于如何将请求分配到服务器某个节点,则需根据自适应动态负载均衡算法的特点,需基于各节点的实时负载情况和服务器硬件性能进行分配算法逻辑的更新。

对每个节点 $S_i \in \{S_1,S_2,S_3\cdots S_n\}(n > 1)$,使用 $W(S_i)$ 表示第 S_i 个节点的默认初始权重。初始默认权值大小由节点初始硬件性能指标决定。采用负载监控采集模块收集服务器各个节点的负载情况,$SL(S_i)$ 负载评价函数表示节点负载情况。权值频繁更新易造成服务器集群抖动,准确的权值调整会使集群负载更均衡,进一步增强集群稳定性。在此引入节点资源利用率 P_i,其定义如下:

$$P_i = \frac{U_i}{W(S_i)} \tag{6}$$

其中,U_i 是 S_i 节点根据 $SL(S_i)$ 函数的实时负载情况,$W(S_i)$ 是当前节点的权值。负载均衡度 σ 可以很好地反映当前服务器集群负载均衡情况,σ 值越小,集群负载均衡程度越好;反之,则集群负载越不均衡,需重新调整权值大小,负载均衡度 σ 定义如下:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - P_{\text{avg}})^2} \tag{7}$$

其中,P_{avg} 表示集群资源利用率的平均值,计算如下:

$$\lambda = P_{\text{avg}} \times k_1 + M \times k_2 \tag{8}$$

式中:M 表示集群离散程度,计算公式为:$M = M_{\max} - \sigma$;$M_{\max}$ 由集群中服务器节点综合负载的最大离散值得出,计算公式为:$M_{\max} = \max\{P_i \mid i = 1,2,\cdots,n \mid\} - P_{\text{avg}}$。相关系数 $k_1 + k_2 = 1$。因此,负载离散阈值 λ 范围在(0,1],从而根据 λ 反映出当前服务器集群负载离散状态。λ 越大,说明当前服务器集群负载离散度越大,反之则说明服务器集群负载离散度较低。

(3)设置负载监控采集模块来收集节点的负载信息。负载监控采集模块通过对服务器节点上在时间段 T 内的信息收集,用以计算负载均衡阈值 λ 是否超过设定阈值,当本次计算的 λ 小于设定阈值,则将不调整节点的权值,反之则调整权值大小,以此来避免集群抖动,从而增强集群的负载均衡。当调整权值时,服务器集群将服务器节点的负载评价指标和初始权重作为依据。本文将自适应算法设计成动态加权轮询算法,$W_{\text{de}}(S_i)$ 表示第 S_i 个节点的默认初始权重,根据服务器节点的硬件性能指标决定,$W(S_i)$ 表示第 S_i 个节点的剩余负载权重即第 S_i 节点的实时权重。权重越大,则该节点被分配到的请求的概率越大。

节点的实时权重根据 $W(S_i)$ 和负载评价函数计算如下:

$$W(S_i) = \frac{W_{\text{de}}(S_i)(1 - U_i)}{\sum_{i=1}^{n} W_{\text{de}}(S_i)} \tag{9}$$

在高负载情况下,使用动态加权轮询的负载均衡算法,考虑了当前节点的负载评价指标与实时性能利用率及阈值,使其分配更加合理,负载更均衡,防止集群抖动。

1.3 基于改进布谷鸟算法的转换阈值计算

本文根据文献研究和进行相应实验对比后发现,Nginx 作为后端服务器集群在高并发、高负载状态下,动态加权轮询负载均衡算法有着较明显的优势;而在低负载状态下,由于服务器在负载调度、节点信息收集等方面会消耗一定资源。这类资源在高并发、高负载状态下,资源的消耗微不足道,但是在低负载情况下,占用了相对较大的资源空间。而静态负载均衡算法因其算法分配逻辑简单,对资源消耗较少,对于集群在低负载状态下有着更良好的适用效果。因此,需计算静态负载均衡算法适用区间到动态负载均衡算法并发量的负载值,即转化阈值 T。本文对布谷鸟算法进行研究分析后,提出一种基于改进的自适应布谷鸟算法,将该算法作用于静动态转换阈值的计算。

布谷鸟算法作为元启发式智能搜索算法,通过模糊逻辑推理,将搜索空间划分为若干个子空间,然后在每个子空间中进行搜索,最终找到最优解[13]。其基本原理是通过模拟布谷鸟鸟巢寄生繁衍,采用一个随机生成的种群,利用列维飞行机制(Levy Flight)有效地求解最优化问题。其流程和计算步骤如下:

(1)初始化布谷鸟种群,设置最大迭代次数、种群规模、搜索空间维度,随机产生 N 个鸟巢,并根据适应度函数计算,将布谷鸟及寄生的鸟巢位置映射

成种群空间中的解 $F(S_i)=(f_1,f_2,\cdots,f_N)$，同时记录最优解 F_{best}。

(2)全局利用列维飞行进行解的位置更新，获取新的候选解位置，利用贪心选择策略保留更好的解。通过适应度函数计算对比，获得当前最优适应度的解 F_{best}，从而计算出静动态转换阈值。

(3)完成全局探索后，在当前解的基础上，采用偏好随机游走的方式完成局部搜索。根据可寄生的鸟巢数量 N 为固定值，鸟巢被宿主发现的概率为 $Pa\in[0,1]$，Pa 的取值为 0.25[14]，随机生成一个随机数 r，通过 r 与 Pa 的比较，若 Pa 小于 r，意味该解为较差解，则产生新的解替代被宿主发现所丢弃的解。

(4)若满足终止条件，则输出 F_{best}；否则，继续进行迭代。

布谷鸟算法基于 Levy 飞行[15]的公式，式(10)：

$$x^{t+1}=x_i^t+r\times\alpha\times\text{Levy}(\lambda)\times(x_{\text{best}}^t-x_i^t) \tag{10}$$

其中，α 为列维飞行的步长，$r\in[0,1]$ 的均为随机数，其中 x_i 表示第 i 个布谷鸟在第 t 代时选择的寄生巢的位置，x_{best} 表示当前最优解。布谷鸟算法通过列维飞行方式，使得算法具有很强的全局搜索能力，以达到获得全局最优解的目的。然而，由于自身存在早熟收敛，优化精度不高从而易陷入局部最优[16]。基于此问题，本文选择了一种基于种群分布熵对步长控制因子 α 进行改进，熵为某一系统体系混乱程度的度量，种群熵为算法寻优过程中种群分布混乱程度的度量，S_j 表示布谷鸟种群在寻优搜索空间中的分布熵，表达式如下：

$$S_j=-\sum_{i=1}^{K}\frac{n_i}{N}\ln\frac{n_i}{N} \tag{11}$$

将空间分为 K 个搜索空间，每个空间含有 $n_i(i=1,2,\cdots,K)$。S 越大，则种群越分散，表明算法未收敛，需要较大的更新步长进行全局搜索；S 越小，则种群越集中，表明算法开始收敛，需要较小的更新步长进行局部搜索。对 α 步长因子进行改造，表达式如下：

$$\alpha=\theta\cdot\frac{S_j-S_{\min}}{2(S_{\max}-S_{\min})} \tag{12}$$

其中，$S_{\max}$、$S_{\min}$ 分别为分布熵理论最大值、最小值，θ 为调整参数。计算步长控制因子的取值范围为 $\alpha\in[0,\theta\cdot\frac{\ln K}{2\ln N}]$。令 α 在 $[0,1]$ 区间内变化，则 $\theta=\frac{\ln K}{2\ln N}$。然而，当全部种群聚集在同一区域内，意味着分布熵 $S=0$，则令步长控制因子 $\alpha=\frac{1}{K}$，使得算法无法再继续更新。通过分布熵这种非线性变化自适应的方式，使得种群熵也随之变化(从大到小)，代表了种群在寻优过程中，能以较快的收敛速度收敛，同时不易陷入局部最优，更大概率获得全局最优解。

根据负载均衡算法及改进布谷鸟算法的目的，来确定适应度函数。本文根据 1.1 节中的服务器节点负载评价指标 $SL(S_i)$ 作为适应度函数的选择标准，使用 1.2 节中的动态负载均衡算法作为方法之一；根据文献[17]，引入最小熵原理，最终确定适应度函数，其式如下：

$$F(S_i)=\sum_{i=1}^{N}\frac{\partial U}{\partial t}\cdot\frac{k(U_i-U_0)}{U_0}-\sqrt{\sum_{i=0}^{N}W(S_i)^2} \tag{13}$$

其中，U_i 是节点 S_i 的服务器节点负载指标；U_0 是服务器集群根据初始硬件负载指标；k 为热力学常量；$W(S_i)$ 是节点 S_i 根据计算所得权值。

根据适应度函数的构建得出目标优化函数，通过寻找目标优化函数的零值解，意味着找到种群的最优解，从而计算出服务器动静态算法转化阈值 T。

本文通过改进布谷鸟算法对服务器集群负载评价指标的适应度函数的计算，使布谷鸟种群跳出局部最优解，朝全局最优解的方向迭代计算，最终找出全局最优解，即找出服务器集群从静态负载均衡算法转化为动态负载均衡算法的最优阈值 T。

1.4　总动静态负载均衡算法

基于熵权法计算指标权重及布谷鸟算法的改进与运用等，提出一种基于改进布谷鸟算法和 Nginx 平台的动静态结合的负载均衡算法，其关键点在于找出动静态转化的最优阈值。通过服务器设置负载监控采集模块，依据所监控的服务器集群负载情况信息，判断是否小于转化阈值 T。若小于，则采用静态加权轮询算法；否则，则采用自适应动态加权轮询算法。同时，根据监控服务器集群的各个节点的负载信息，若负载变化大于负载均衡阈值 λ，则改变权值大小，反之则不改变。从而来获得更好的服务器集群负载均衡效果。算法流程如图 1 所示。

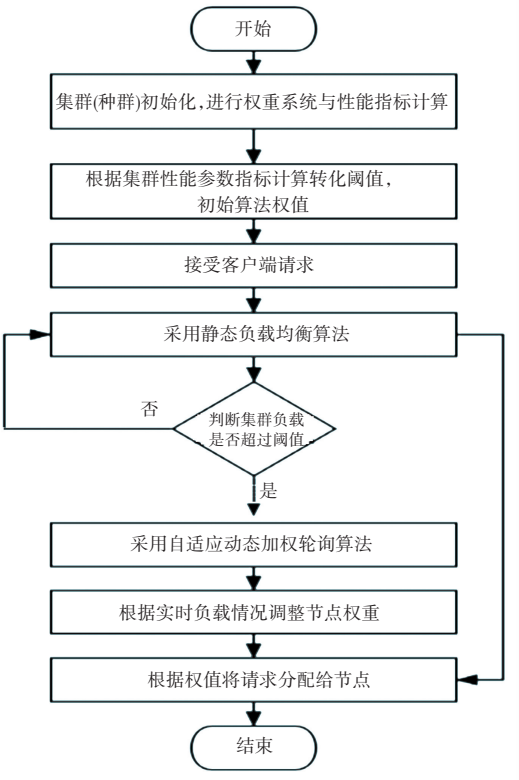

图 1 算法流程

Fig. 1 Algorithm flow chart

2 实验和结果分析

2.1 实验环境

本文通过在虚拟机中搭建八台服务器,其中一台作为客户端测试连接,用于某学工系统;一台作为 Nginx 反向代理服务器;其余 6 台作为后台服务器,从而达到搭建服务器集群系统的目的。实验环境相关参数见表 1。

表 1 实验主要参数配置

Table 1 Main parameter configuration of the experiment

服务器类型	CPU/GHz	内存/G	IO/(MB·s^{-1})	带宽/M	数量
客户端	3.6	8	400	50	1
反向代理服务器	3.6	4	400	50	1
后端服务器	3.6 2.8	8 4	400 300	50 20	3 3

本文采用 Siege 性能压测工具,用 PTS 模拟出大量用户访问业务的真实场景,同时进行各个服务器节点的数据采集,服务器节点每隔 4 s 上传一次节点负载情况。为检测算法性能,采取并发模式进行压测,设置每次不断增加 10%-20%的并发请求数。

2.2 实验结果分析

本文算法结合静态负载均衡算法和动态负载均衡算法,因此实验中分别对 Nginx 默认加权轮询算法(WRR 算法)和最小连接数算法(least-con),亦选取文献[8]的动态权重算法进行对比。通过文献和资料收集,本文采取在不同并发请求数下的平均响应时间和实际连接数评价算法性能优劣的指标。进行 10 次实验,得到各个算法的平均响应时间和实际连接数,并且对实验数据取均值,测试结果如图 2、图 3 所示。

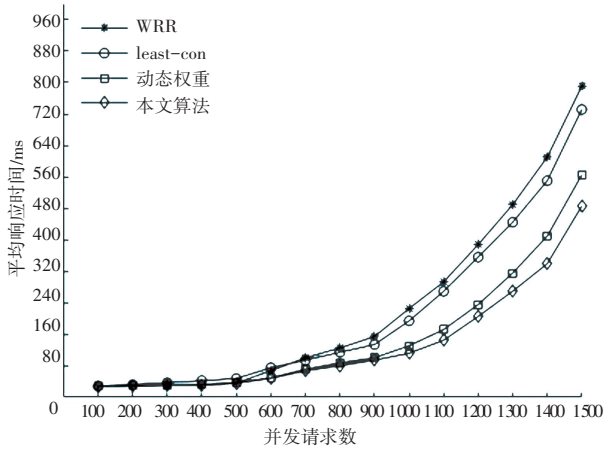

图 2 算法响应时间与并发数关系图

Fig. 2 Relationship between algorithm response time and concurrency number

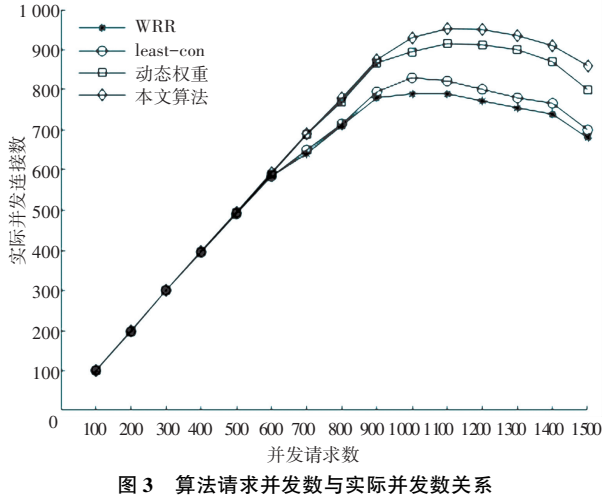

图 3 算法请求并发数与实际并发数关系

Fig. 3 Relationship between algorithm concurrent requests and actual concurrent numbers

从图 2 可以看出,算法平均响应时间随着并发数的增加而增长。当并发数在 700 之前,WRR 和本文算法相对响应时间较短,各算法响应时间没有太大区别。并发数在 700 以后,WRR 和 least-con 算法响应时间开始随着并发数的增大而大幅增长。并发数在 1 100 以后,动态权重和本文算法开始大幅

增长，但相较于 WRR 算法和 least-con 算法，仍有着明显优势。在并发数为 1 500 时，相较于动态权重响应时间降低了 18%。总体来看，本文算法在高并发情况下的响应时间相比其他算法更低、时间最短，表明本文算法在高并发情况下更能合理分配请求。

从图 3 可以看出，WRR 算法和 least-con 算法在并发请求数达到 700 以后，实际并发连接数开始丢失；动态权重算法和本文算法在并发请求数达到 1 000 以后开始丢失。相较于 WRR 算法和 least-con 算法，本文算法明显高于前二者；相较于动态权重算法，本文算法的并发连接数在 1 000-1 500 的区间下，实际响应数均高于动态权重算法 12%左右。

综上实验结果，表明本文提出的基于改进布谷鸟算法的 Nginx 负载均衡算法在低负载情况下，有着更低的平均响应时间；在高并发、高负载情况下，相较于 WRR 算法、least-con 算法以及动态权重算法有更良好的实际并发连接数和更低的响应时间，达到了更好的负载均衡效果。

3　结束语

为实现服务器集群在高并发情况下具有良好的负载均衡效果，本文基于熵权法对各个硬件指标权重计算，综合 CPU 性能、IO 性能、内存、带宽等负载评价指标，根据监控服务器节点负载情况，自适应修改节点权值大小。通过改进布谷鸟算法计算动静负载均衡转换阈值，提出一种基于改进布谷鸟算法的动静态结合的负载均衡算法。实验结果表明，本算法能够合理分配请求到服务器节点，有效降低平均响应时间和提高实际并发连接数，提高服务器集群负载均衡效果。

参考文献

[1] 嵇小飞. Web 服务器集群系统的自适应负载均衡调度[J]. 中小企业管理与科技(中旬刊),2016(6):153-154.

[2] 陈大才. 基于 Nginx 的高并发访问服务器的研究与应用[D]. 沈阳:中国科学院沈阳计算技术研究所,2018.

[3] 刘佳祎,崔建明,智春. 基于 Nginx 服务器的动态负载均衡策略[J]. 桂林理工大学学报,2020,40(2):403-408.

[4] Kashif Hussain, Mohd Najib Mohd Salleh, Yuli Adam Prasetyo, et al. Personal best cuckoo search algorithm for global optimization [J]. International Journal on Advanced Science, Engineering and Information Technology,2018,8(4):1209-1217.

[5] 葛钰,李洪赭,李赛飞. 一种 web 服务器集群自适应动态负载均衡设计与实现[J]. 计算机与数字工程,2020,48(12):3002-3007.

[6] Harikesh Singh, Shishir Kumar. Dispatcher based dynamic load balancing on web server system [J]. International Journal of System Dynamics Applications (IJSDA),2012,1(2):15-27.

[7] 谭畅,谭歆,胡磊,等. 云中心基于 Nginx 的动态权重负载均衡算法[J]. 重庆邮电大学学报(自然科学版),2021,33(6):991-998.

[8] 李慧斌,何利力. 基于预测阈值的动态权值负载均衡算法[J]. 软件导刊,2020,19(6):85-89.

[9] 胡逸飞,包梓群,包晓安. 一种基于改进遗传算法的动静态负载均衡算法[J/OL]. 电子科技:1-7[2023-03-01].

[10] 傅文渊. 具有万有引力加速机理的布谷鸟搜索算法[J]. 软件学报,2021,32(5):1480-1494.

[11] 李坤. 基于动态反馈机制的服务器负载均衡算法研究[J]. 电子科技,2015,28(9):45-49.

[12] WEI Xing, YANG Hua, HUANG Wentao. A genetic-algorithm-based optimization routing for FANETs [J]. Frontiers in Neurorobotics,2021,15:697624.

[13] 张燕,袁书卷. 基于信息熵的布谷鸟搜索算法[J]. 西安文理学院学报(自然科学版),2022,25(4):17-22,36.

[14] WANG Wen, XIE Cong. A cuckoo search algorithm based on self-adjustment strategy[J]. Journal of Physics: Conference Series, 2018,1087(2):022003.

[15] ALHAIDARI F, BALHARITH T Z. Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling [J]. Computers,2021,10(5):63.

[16] 刘洪达,李德全,王栋. 基于种群熵的变步长布谷鸟搜索算法[J]. 计算机仿真,2022,39(9):370-376.

[17] 高晶,李元香,纪道敏,等. 基于最小熵产生选择策略的遗传算法研究[J]. 计算机应用与软件,2019,36(10):238-244,304.