

基于ESXi性能计数的Nginx负载均衡算法研究

潘 谜¹, 李 旺²

(1. 集美大学理学院, 福建 厦门 361021; 2. 集美大学计算机工程学院, 福建 厦门 361021)

[摘要] 在现有Nginx负载均衡算法的基础上, 提出基于ESXi性能计数的负载均衡算法(dynamic weight based on ESXi performance counter, DWEPC), 旨在通过均衡集群中的各节点负载, 达到降低节点内资源过载的情况, 从而提高系统性能。试验表明, 相比于Nginx内置均衡算法, DWEPC算法具有更优的效果。

[关键词] Nginx; ESXi性能计数; 负载均衡; DWEPC

[中图分类号] TP 391.9

Research on Nginx Load Balancing Algorithm Based on ESXi Performance Counter

PAN Mi¹, LI Wang²

(1. School of Science, Jimei University, Xiamen 361021, China;

2. Computer Engineering College, Jimei University, Xiamen 361021, China)

Abstract: Based on the existing Nginx load balancing algorithm, this paper proposes a load balancing algorithm named DWEPC(dynamic weight based on ESXi performance counter), which aims to reduce the resource overload in single node by balancing the load of each node in the cluster, and improves the system performance thereby. Experiments show that the DWEPC algorithm has better results than the Nginx built-in equalization algorithm.

Keywords: Nginx; ESXi performance counter; load balancing; dynamic weight based on ESXi performance counter

0 引言

Nginx^[1]是一款开源的轻量级Web服务, 提供高性能的HTTP/HTTPS服务。Nginx因其具稳定性、低资源消耗、高性能等特点, 近年来广受主流网络服务商的青睐。百度、京东、淘宝、新浪、腾讯等均使用Nginx或其扩展作为Web服务。目前, 对Nginx的研究主要集中在两个方面: 一个是模块扩展(如淘宝的Tengine), 另一个是反向代理的负载均衡及并发研究。关于Nginx负载均衡的研究, 主要有两个方向: 一类是基于非反馈的均衡调度算法; 另一类是基于反馈的均衡调度算法。基于非反馈式的均衡调度算法主要有Nginx内置的轮询算法(round robin)、基于权重的调度算法(Weight)、IP散列调度算法(IP hash)、URL散列调度算法(URL hash)等, 这些算法简单、资源消耗少, 但在一定条件下会导致调度严重失衡。基于反馈式的均衡调度算法是根据集群中各节点实际负载情况, 决定下

[收稿日期] 2019-08-03

[基金项目] 福建省科技厅重点项目(2017H0026); 福建省教育厅科技项目(JT180285); 校基金及教改项目(ZC2018031、JY18082)

[作者简介] 潘谜(1978—), 女, 讲师, 从事概率论与数理统计、算法分析与设计方向研究。

一步的调度策略, 目前主要集中在集群各节点权重及实际负载的计算算法的改进, 如 Nginx 的加权最小连接调度算法 (weighted least - connection scheduling)^[1]、动态反馈负载均衡算法^[2]、动态自适应负载均衡算法^[3]、自适应和预测的轮询算法^[4]、基于 HAProxy 的轮询算法^[5]、云环境下自组织感知体均衡算法^[6]。一般算法中使用的评价参数包括连接数、CPU 使用率、内存使用率、I/O 使用率、网络使用率、请求响应时间等^[7-9], 根据这些参数动态计算权重, 用于决定下一次请求的分配策略。在一定程度上, 这些改进后的算法能够提高部分系统性能, 但仍存在不足之处: 1) 评价参数需要由各节点计算并收集, 这不可避免地增加了系统开销, 且异构操作系统兼容性也会增加模块升级及维护难度; 2) 不均衡任务分配策略中仅仅考虑异构参数的同一化权重, 在极端情况下, 有可能导致瞬时节点瓶颈, 甚至导致节点崩溃。

ESXi 是 VMware vSphere^[10] 的核心组件, 提供虚拟化服务, 是专门为运行虚拟机、最大限度降低配置要求、简化部署而设计的。它具有可靠、安全、简化部署和配置、减少管理开销等特性。ESXi 提供的功能有镜像生成器 (image builder)、面向服务的无状态防火墙、主机硬件全面监控、虚拟机迁移 (vMotion)、安全系统日志 (secure syslog)、VMware vSphere Auto Deploy、扩展增强型 Esxcli 框架, 以及新一代的虚拟机硬件^[11]。ESXi 扩展了 SNMP V3 支持, 实现主机硬件资源的全面监控, 同时提供 VMware API^[12], 支持多种编程环境, 为主机管理、自动化部署与管理提供接口。ESXi 性能计数是 VMware API 提供的功能之一, 包括主机和虚拟机 CPU、内存、网络、磁盘等统计量。

针对上述问题, 本文引入 ESXi 性能计数 (ESXi performance counter, EPC), 提出基于 ESXi 性能计数的均衡算法 (dynamic weight based on ESXi performance counter, DWEPC), 旨在使用近似实时的性能指标, 预测并指导均衡分配策略, 降低均衡模块的复杂度及各节点的通信开销, 从而达到提升性能的目的。

1 负载均衡算法

Nginx 反向代理主要由 upstream 模块实现。它处理用户请求的一般步骤如下: 1) 反向代理接受用户的请求; 2) 调用负载均衡算法, 获取可调度的子节点索引值; 3) 将用户请求转发给步骤 2) 中返回的子节点, 并等待响应; 4) 转发子节点响应给用户。其中, 负载均衡调度算法为反向代理提供当前请求转接的子节点索引, 它的优劣直接影响调度的性能。

1.1 权重更新算法

对于 n 个节点的 Nginx 集群, 本研究选取 CPU、内存、网络、磁盘 I/O 四个指标作为负载均衡算法调度指标。对于节点 $i(i = 1, 2, \cdots, n)$, 其固有性能表示为 $c_i = (c_{i1}, c_{i2}, c_{i3}, c_{i4})$, 代表物理资源的最大限值。则 n 个节点固有性能使用矩阵表示为: $C = (c_{ij})_{n \times 4}$ 。

由于任何反馈式的性能计数都是离散的 (这是综合考虑性能及信息采集的特点而定的), ESXi 性能计数也是离散的。考虑最近 m 个采集的计数 (当采集数量超过 m 时, 舍弃开头的数据, 仍然只保留 m 个窗口大小), 每次采集的计数表示为 n 行 4 列矩阵,

$$P_j = \begin{pmatrix} p_{11}^j & p_{12}^j & p_{13}^j & p_{14}^j \\ \vdots & \vdots & \vdots & \vdots \\ p_{n1}^j & p_{n2}^j & p_{n3}^j & p_{n4}^j \end{pmatrix}, j = 1, 2, \cdots, m, \tag{1}$$

定义 \bar{P}_j 为 P_j 的标准化矩阵, 如式 (2) 所示:

$$\bar{P}_j = \begin{pmatrix} p_{11}^j/c_{11} & p_{12}^j/c_{12} & p_{13}^j/c_{13} & p_{14}^j/c_{14} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n1}^j/c_{n1} & p_{n2}^j/c_{n2} & p_{n3}^j/c_{n3} & p_{n4}^j/c_{n4} \end{pmatrix} \overset{\text{记为}}{=} \begin{pmatrix} \bar{P}_{11}^j & \bar{P}_{12}^j & \bar{P}_{13}^j & \bar{P}_{14}^j \\ \vdots & \vdots & \vdots & \vdots \\ \bar{P}_{n1}^j & \bar{P}_{n2}^j & \bar{P}_{n3}^j & \bar{P}_{n4}^j \end{pmatrix}, \tag{2}$$

其中, \bar{P}_j 矩阵所有元素取值均在区间 $[0, 1]$, 表示资源使用率。

调度算法中,记录分配到各节点的次数,每次产生新的 ESXi 性能计数点时,保存次数向量作为上一计数点的调度历史向量。仍然只考虑最近 m 个采集计数,记录节点调度次数为: $V_j = (v_{1j}, v_{2j}, \dots, v_{nj})^T, j = 1, 2, \dots, m$ 。定义权重向量 W , 各分量分别表示 4 种性能指标的权重: $W = (w_1, w_2, w_3, w_4)^T$ 。定义 Ψ 为: $\Psi = \sum_{j=2}^m |\bar{P}_j W - V_{j-1}|^2$, 为了使各节点负载均衡, 则必须使 Ψ 最小, 也就是 Ψ 对 4 个权重参数的偏导数为 0, 即:

$$\partial \Psi / \partial w_i = 0, i = 1, 2, 3, 4. \quad (3)$$

解公式 (3) 所示的方程组得到向量 W , 即为当前权重向量。

由于性能计数是每隔一段时间采集一次, 因此权重向量也是每隔一段时间更新一次, 与性能计数同步。

1.2 主调度算法

由于计数并非实时采集, 调度算法必须维护并预测最后一次采集后的各节点的调度情况和近似负载。对于上述 m 个采集计数, 考虑最后一次 (第 m 次) 的标准化矩阵 \bar{P}_m 和当前调度次数向量 V_m 。选取 α 为 $(0, 1]$ 之间的常量, 表示各节点预期的最大负载率, 一般情况下取 $0.9 \sim 1.0$ 。主调度算法定义如下:

输入: 性能计算标准化矩阵 \bar{P}_m 、权重向量 W 、调度向量 V_m 。

1) 计算矩阵 $Q = (q_{ij})_{n \times 4}$, 其中 $q_{ij} = \alpha - \bar{p}_{ij}^m, 1 \leq i \leq n, 1 \leq j \leq 4$, Q 表示各节点各项资源剩余率矩阵;

2) 计算 $L = QW - V_m$, L 表示各节点剩余可分配次数向量;

3) 在向量 L 中寻找最大元素对应的行标 k (若多行同时最大, 则随机选取一行);

4) 更新 V_m 的第 k 行元素的值: $v_{km} = v_{km} + 1$ 。

输出: k 。

DWEPC 算法的返回值表示待分配的子节点索引, 反向代理根据该索引值选择请求转接的子节点。

1.3 均衡指标

在含有 n 个节点的集群中, 调度时采集了 N 个性能计数, 则每个节点 $i (i = 1, 2, \dots, n)$ 的性能指标 $k (k = 1, 2, 3, 4)$ 的均值的计算公式为: $E(i, k) = \sum_{j=1}^n \bar{p}_{ik}^j / n$ 。整体均值为各个节点均值的平均数 $E(k) = \sum_{i=1}^n E(i, k) / n$ 。均和方差定义为: $\delta^2(k) = \sum_{j=1}^N \sum_{i=1}^n (\bar{p}_{ik}^j - \sum_{q=1}^n \bar{p}_{qk}^j / n)^2 / N$ 。均和方差 $\delta^2(k)$ 本质上是同一时刻各节点负载率的差异值, 是衡量均衡调度效果的重要指标。

2 试验结果

2.1 测试环境

测试环境搭建于 VMware ESXi 6.5 Update 1 平台上, 双路 Intel Xeon L5630, CPU 主频 2.13 GHz (4 核 8 线程), 内存 40 GB。搭载 Nginx 的虚拟机节点配置有双核 CPU、2GB 内存、40GB 硬盘 (精简置备)、VMXNET3 网卡 (虚拟万兆网络适配器)。

提供 Nginx Web 服务的主虚拟机节点安装 Nginx - 1.17.1, 其余每个虚拟机子节点安装 Nginx - 1.17.1、Tomcat 9.0.21。

测试时, 取 n 为 5, 即包含 5 个子节点, 每个子节点中运行 3 个 Tomcat 实例, 共 15 个 Tomcat 实例。其中, 主节点 worker 进程数设置为 8, 子节点设置为 4。

由于实际应用时, CPU、网络资源尤其关键, 它的性能直接影响到其他几个指标, 因此本文设计的测试用例也主要针对 CPU、网络的调度。测试用例 T1 的内容为 5 个静态页面, 大小分别为 1 KB、

16 KB、64 KB、512 KB、1 MB; 测试用例 T2 的内容为 5 个动态页面, 页面中分别包含 50^k 次加法运算, $k = 1, 2, \dots, 5$ 。

为模拟现实操作, 每个页面请求之间间隔 2 s。测试用例 T1 主要针对 CPU 和网络负载, 而 T2 则主要测试 CPU 负载均衡。

2.2 试验结果及分析

选取 Nginx 最具代表性的 Weight 均衡策略, 与本文提出的 DWEPC 算法 (取 $m = 20$, 采集时间间隔为 3 s) 对比, 在 5 个子节点的环境下, 分别运行 T1 和 T2 测试用例。性能测试终端使用 Apache JMeter, 虚拟用户数 (virtual user, 简写 VU) 分别取 100、500、1000、1500、2000, 运行 10 min, 最终每个节点采集 N 次 (N 由运行时长和采集时间间隔确定)。

由于测试用例对于内存需求趋于常量, 且 Nginx 本身拥有页面缓存模块, 使得磁盘访问也基本可以忽略不计。本文仅列出 CPU、网络的测试结果, 在不同虚拟用户数 (VU) 下分别运行 3 次取平均值, 所得的均和方差如表 1 所示 (为了方便数值比较, 这里取 $N \delta^2(k)$ 为均和方差)。

表 1 不同 VU 下的均和方差
Tab. 1 Sum variance at different VUs

测试内容		算法	VU				
Test	Algorithm		100	500	1000	1500	2000
T1	CPU	Weight	0.002	0.009	0.021	0.078	0.152
		DWEPC	0.004	0.013	0.020	0.021	0.141
	网络	Weight	0.986	22.775	61.669	251.165	406.259
		DWEPC	2.267	19.732	64.550	78.197	180.012
T2	CPU	Weight	0.886	3.823	0.561	0.629	0.327
		DWEPC	0.247	4.961	0.008	0.003	0.021
	网络	Weight	1×10^{-6}	6×10^{-5}	2×10^{-4}	3×10^{-4}	3×10^{-4}
		DWEPC	1×10^{-5}	0.002	0.001	0.001	0.001

从表 1 可以看出, 两种算法在 T1 测试的 CPU 均衡、T2 测试的网络均衡调度上性能相当, 但在 T1 的网络均衡、T2 的 CPU 均衡调度上, DWEPC 总体上拥有更好的性能 (见图 1、图 2)。

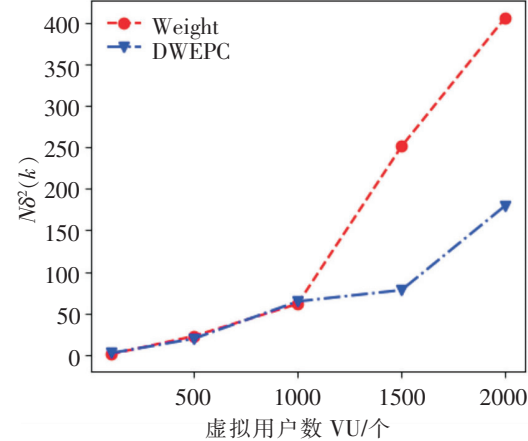


图 1 Weight 与 DWEPC 在 T1 的网络均衡对比

Fig.1 Comparison of Weight and DWEPC network balance at T1

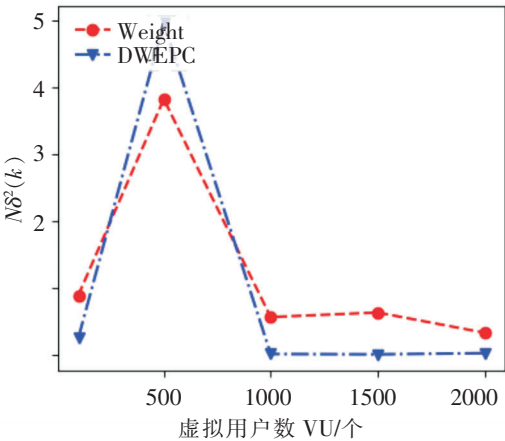


图 2 Weight 与 DWEPC 在 T2 的 CPU 均衡对比

Fig.2 Comparison of Weight and DWEPC CPU balance at T2

进一步, 考察两种算法在 T2 测试下, 虚拟用户数 (VU) 为 1000 时, 某次运行时各节点 CPU 实际负载走势, 分别如图 3、图 4 所示。图 3 中, 节点 5 的 CPU 负载几乎都在 100% 附近, 而其他几个节点却都在 0~20% 之间, 即, 节点 5 几乎承担了所有的 CPU 运算工作。图 4 中, 所有节点的 CPU 负载几乎都在变化, 没有出现某一节点过载的情况, 因此更能合理利用各节点资源, 从而提升整体吞

吐率。尽管图 4 看起来杂乱无章，但那是因 T2 测试用例中的页面极端不平衡导致的，如，第 5 个页面是第 1 个页面计算量的 6.25×10^6 倍。

由此可见，DWEPC 算法能够根据 ESXi 性能计数，更均衡地实现各节点负载调度。

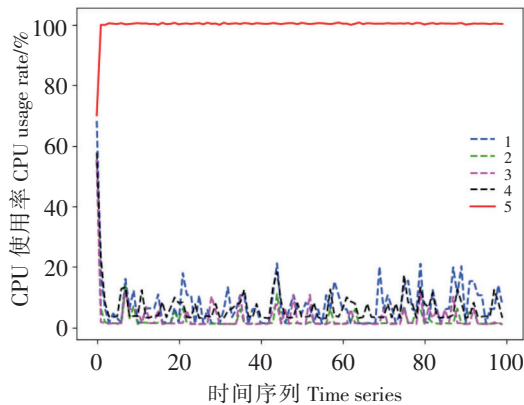


图 3 Weight 算法下各节点 CPU 负载

Fig.3 CPU load of each node under Weight

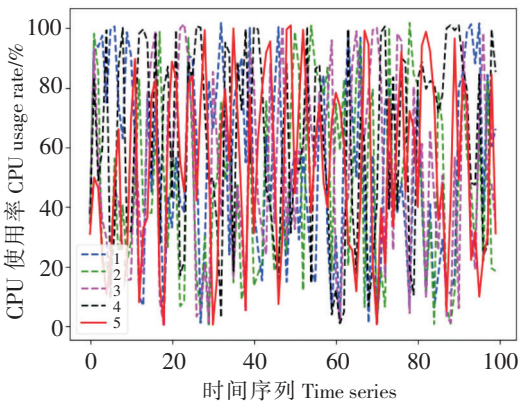


图 4 DWEPC 算法下各节点 CPU 负载

Fig.4 CPU load of each node under DWEPC

3 结论

本文提出了基于 ESXi 性能计数的 Nginx 负载均衡算法（DWEPC），与 Weight 算法相比，DWEPC 具有更优的节点均衡负载率，避免了节点过载的情况，从而提高系统性能。但，DWEPC 算法还不够完善，其算法本质上是基于均值预测的，对突发改变不够灵敏。

[参考文献]

[1] Nginx. <http://nginx.org> [DB/OL]. [2019-06-25].

[2] 黄静, 李炳. 基于 Nginx 的 Web 服务器性能优化研究 [J]. 浙江理工大学学报 (自然科学版), 2016, 35(4): 600-606.

[3] 张云, 许江淳, 李玉惠, 等. 基于 Nginx 服务器负载均衡技术的研究与改进 [J]. 软件, 2017, 38(7): 6-12.

[4] ALAM F, THAYANANTHAN V, KATIB L. Ananysis of round-robin load-balancing algorithm with adaptive and predictive approaches [C] //2016 UKACC 11th International Conference on Control (CONTROL), Belfast: UKACC, 2016: 1-7. DOI:10.1109/CONTROL.2016.7737592.

[5] PROMONO L H, BUWONO R C, WASKITO Y G. Round-robin algorithm in HAProxy and Nginx load balancing performance evaluation: a review [C] //2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia: IEEE, 2018: 367-372.

[6] FAUSTINA J M, PAVITHRA B, SUCHITRA S, et al. Load balancing in cloud environment using self-governing agent [C] //Proceedings of the Third International Conference on Electronics Communication and Aerospace Technology (ICECA 2019), Coimbatore, India: IEEE, 2019: 480-483.

[7] 张娜, 马琳. 基于 Nginx 负载均衡的动态分配技术研究 [J]. 齐齐哈尔大学学报 (自然科学版), 2019, 35(1): 27-30.

[8] 戴伟, 马明栋, 王得玉. 基于 Nginx 的负载均衡技术研究与优化 [J]. 计算机技术与发展, 2019, 29(3): 77-80.

[9] 毛正雄, 赵志宁, 孙北宁. 基于 Nginx 的 Web 响应加速优化研究 [J]. 自动化与仪器仪表, 2018(4): 31-35.

[10] VMware. <https://www.vmware.com> [DB/OL]. [2019-07-01].

[11] 马博峰. VMware、Citrix 和 Microsoft 虚拟化技术详解与应用实践 [M]. 北京: 机械工业出版社, 2013: 18.

[12] VMware API and SDK Documentation. https://www.vmware.com/support/pubs/sdk_pubs.html [DB/OL]. [2019-07-01].

(责任编辑 朱雪莲 英文审校 黄振坤)