doi:10.3772/j.issn.1002-0470.2020.08.005

基于二部图最大匹配的动态负载均衡算法①

周 磊②* 孟利民③* 周立鹏* 蒋 维**

(*浙江工业大学信息工程学院 杭州 310023)

(**浙江树人大学信息科技学院 杭州 310015)

摘 要 负载均衡算法是服务器集群的核心技术之一,其关键在于如何将任务均衡地分配至各个服务器,为此提出了一种基于二部图最大匹配的动态负载均衡算法。该算法以服务器所执行任务的任务量与实际完成时间的比值作为服务器的负载指标,并将负载指标实时反馈给集群系统中的管理服务器。根据待分配任务的任务量和期望完成时间以及各个服务器的负载指标,构建服务器与任务的二部图,并采用 Edmonds 的匈牙利算法求解二部图的最大匹配,最后按匹配结果将任务实时发送至对应服务器。实验结果表明,该算法拥有较好的负载均衡效果且更快地完成所有任务。

关键词 动态负载均衡;任务分配;服务器集群;二部图;最大匹配

0 引 言

随着网络技术的高速发展,互联网服务已经成为日常生活中不可或缺的部分。由于互联网用户的爆发式增长,服务器常常在短时间内收到大量并发的用户请求,若不能及时处理这些请求将影响用户体验,降低服务质量。因此,单个服务器远远无法满足大量的服务请求,联合多个服务器的服务器集群应运而生。如何合理地分配集群服务器的任务并满足最大的服务需求是服务器集群需要解决的关键技术问题,而负载均衡是解决服务器集群难点的核心技术之一,能够平衡集群中各个服务节点的负载,充分发挥服务器集群的性能。目前国内外已经提出了各种负载均衡算法[1]。其中有静态负载均衡算法,如轮询调度算法、加权轮询调度算法[2]、目标地址散列调度算法等,这类算法易于实现,但不考虑各个服务器节点的负载状态,容易导致负载不均衡;有动态负载均衡算法[3,4],如最小连接数算法、一致性哈希算法[5]、动态反馈算法[6]等,这类算法没有考虑服务器间的性能差异和任务请求的大小,不能准确地判断服务器真实的负载状态。动态负载均衡算法注重于服务器负载状态的计算和反馈,文献[7]提出一种基于排队论综合指标评估的动态负载均衡算法,采用服务器 M/M/1 排队模型,以任务的平均排队时长和平均等待时长来计算负载,但其计算需花费较多时间,具有一定的延后性。文献[8]提出了一种改进的流媒体集群动态负载均衡调度算法,通过集群节点任务数变化量动态修改服务器负载的反馈周期,并按服务器负载状况进行分类,但当反馈周期过小时,服务器将频繁计算负载,增加服务器压力。在异构网络应用中,文献[9]提出了一种基于模糊控制的负载均衡决策机制,用于避免异构网络中频繁切换。文献[10]提出了一种基于接入选择和业务转移的异构网络动态负载均衡机制,同时提高系统利用率和负载均衡性。还有注重任务调度的遗传算法、模拟退火算法[11]、粒子群算法[12]等,这类算法实现较为复杂,且容易陷入局部最优解。文献[13]提出一种基于遗传算法的负载均衡机制,引

① 国家自然科学基金(61871349)和浙江省基础公益(LY18F010024,LQ19F010013)资助项目。

② 男,1995 年生,硕士生;研究方向:流媒体通信,负载均衡技术等;E-mail:2111703026@zjut.edu.cn

③ 通信作者,E-mail:mlm@zjut.edu.cn

(收稿日期:2019-06-23)

入了投资组合 Mean-Variance 模型来设置服务集群中节点利用率的权重,以最小化任务完成时间为条件来获得最优的权值向量,即为每个服务器的资源利用率分配权值从而得到更有效的组合适应度函数,但其计算效率一般,对服务器的响应时间有一定的影响。

影响负载均衡算法性能的关键在于如何动态地将任务均衡地分配至各个服务器节点,因此本文提出了一种基于二部图最大匹配的动态负载均衡算法。二部图最大匹配[14]是指二部图中不相邻的边组成的集合中含边数最多的集合,常用于解决任务指派、资源分配[15]、数据匹配[16]等问题。在服务器集群系统中,管理服务器按照一定的机制将任务队列中的任务分配给集群中的各个服务器节点,这就是可用二部图最大匹配解决的现实应用场景之一。本文算法以服务器和任务作为图顶点的子集 X 和 Y,先预计任务的任务量和期望完成任务所需的时间,并采用由服务器反馈得到的任务量与任务实际完成时间的比值作为服务器的负载指标[17],若任务预计的任务量与服务器负载指标的比值小于期望完成任务所需的时间,则将该任务顶点与服务器顶点相连作为图的边,由此得到服务器与任务的二部图,再采用 Edmonds 的匈牙利算法[18]求该二部图的最大匹配,然后将任务发送给匹配到的服务器,从而动态地实现任务的调度和服务器的负载均衡。

1　负载均衡算法调度模型

本文提出的基于二部图最大匹配的动态负载均衡算法的系统框架如图 1 所示。整个系统主要由管理服务器和服务器集群组成。其中服务器集群中的服务器用于执行任务,并计算自身的负载指标返回给管理服务器。管理服务器可分为 4 个模块,分别是任务分配模块、任务预估模块、负载指标接收模块和计算模块。负载指标接收模块负责接收服务器反馈的负载指标,并更新当前保存在管理服务器的服务器集群负载信息。任务预估模块负责预计待匹配任务的任务量和期望完成任务所需的时间。计算模块根据各个服务器的负载指标以及任务的任务量和期望完成时间,构建服务器与任务之间的二部图并求其最大匹配。任务分配模块按照由计算模块返回的匹配结果将任务分配给各个服务器。

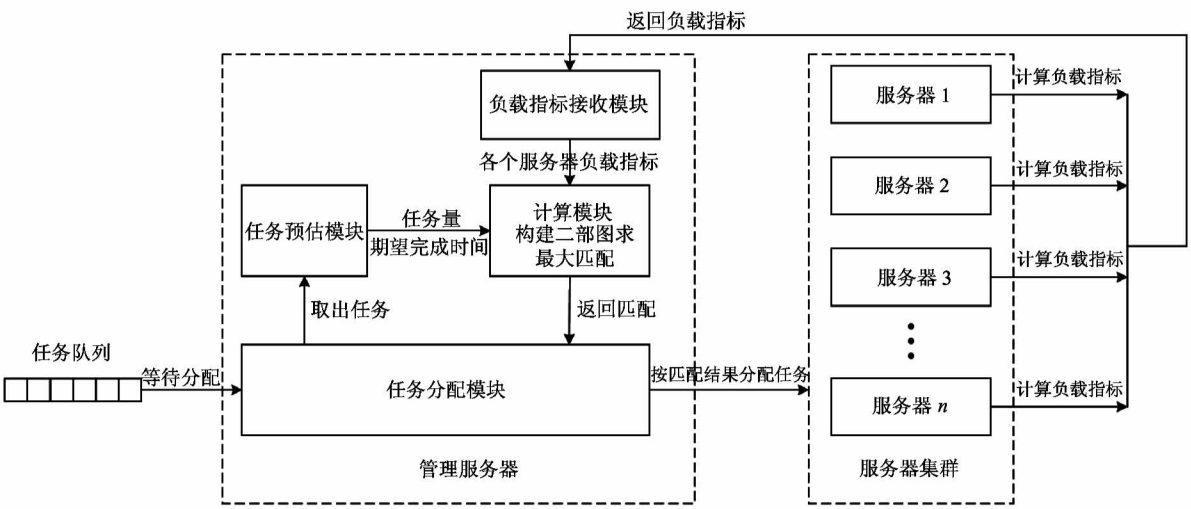


图 1　基于二部图最大匹配的动态负载均衡算法系统框架

服务器每执行一个任务,将记录该任务的任务量 r,任务开始时间 t_{start} 和任务完成时间 t_{end},并由式(1)得到服务器当前负载指标 v。

$$v = \frac{r}{t_{end} - t_{start}} \tag{1}$$

服务器负载指标代表服务器完成任务的速率,与常见算法采用中央处理器(CPU)空闲率、内存空闲率等多参数计算负载指标相比,降低了服务器负载计算的复杂度,且更直接地反映了当前服务器的负载状态。若 v 越大,则说明服务器的负载较低,能

更快地完成任务;若 v 越小,则说明服务器的负载较大,完成任务需要消耗更多的时间。那么服务器集群的负载指标平均值为

$$\bar{v} = \frac{1}{n}\sum_{i=1}^{n} v_i \tag{2}$$

其中,n 为集群中服务器数量。由此可以得到各个服务器负载指标的方差 σ^2:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (v_i - \bar{v})^2 \tag{3}$$

方差 σ^2 越小,说明服务器之间的负载越均衡。

2　负载均衡算法流程

本文算法是基于任务队列、服务器集群和各服务器负载指标构建二部图,并根据二部图的最大匹配结果向各服务器分配任务,其总体流程图如图 2 所示。

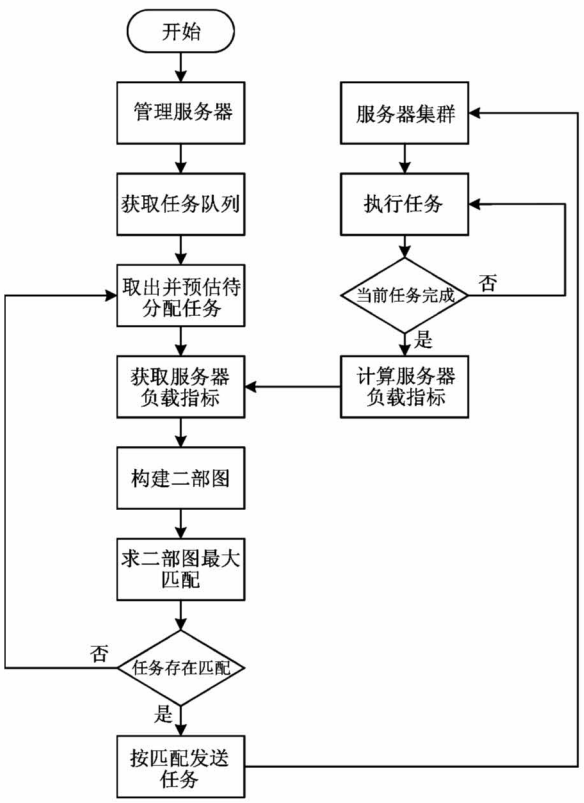


图 2　算法流程图

整个算法流程主要由管理服务器和服务器集群共同工作完成。首先管理服务器中维护着一个任务队列,任务队列中含有一定量待执行的任务,由集合表示为 $J_q = \{J_1, J_2, J_3, \cdots\}$。服务器集群中有 n 台服务器,由集合表示为 $S = \{S_1, S_2, S_3, \cdots, S_n\}$,服务器 S_i 主要负责接收并执行管理服务器为其分配的任务 J_m。服务器 S_i 将在任务 J_m 被执行前后记录该任务的任务量 R_m,以及执行任务的开始时间 T_{mstart} 和结束时间 T_{mend},同时由式(1)计算得到当前服务器 S_i 的负载指标 v_i,并将负载指标 v_i 反馈至管理服务器。

管理服务器的负载指标接收模块中记录着服务器集群中各个服务器的负载指标,由集合表示为 $v = \{v_1, v_2, v_3, \cdots, v_n\}$。各个服务器的初始化负载指标可通过管理服务器向各个服务器发送一个测试任务 J_0 获得,其任务量为单位任务量 R_0,任务复杂度为 C_0,任务复杂度与任务实际计算循环次数相关。

管理服务器记录各个服务器的初始化负载指标后,从任务队列中按顺序取出与服务器数量相等的 n 个任务,由集合表示为 $J = \{J_1, J_2, J_3, \cdots, J_n\}$,同时通过任务预估模块得到对应任务的任务量为 $R = \{R_1, R_2, R_3, \cdots, R_n\}$,期望完成任务所需时间为 $T = \{T_1, T_2, T_3, \cdots, T_n\}$。那么对于任务 J_m,其任务量 R_m 可由式(4)得到:

$$R_m = \frac{C_m}{C_0} R_0 \tag{4}$$

其中,R_0 为单位任务量,C_0 为测试任务的复杂度,C_m 为任务 J_m 的复杂度。任务 J_m 的期望完成时间 T_m 可由式(5)得到:

$$T_m = \frac{R_m}{\alpha R_0} T_0 \tag{5}$$

其中,T_0 为测试任务实际运行时间,α 的取值为经验值。

管理服务器通过任务预估模块得到任务量 R 和期望完成任务所需时间为 T 后,根据实时记录的服务器负载指标 v,开始构建二部图。一般的图可由 $G = \{V, E\}$ 表示,其中 V 为图 G 的顶点集合,E 为图 G 的边集合,由此管理服务器以服务器集合 S 和任务集合 J 作为图的顶点集可得无边图 G,如图 3 所示,显然 $V = S \cup J$, $E = \varnothing$。

然后管理服务器对所有服务器和任务进行遍历,对于任意一个服务器 S_i 和一个任务 J_m,可由

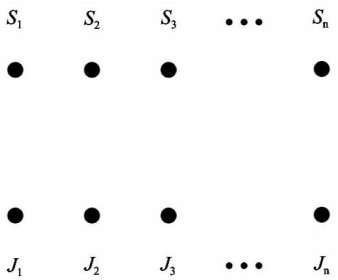


图 3　以集合 S 和集合 J 为顶点的无边图 G

式(6)预估服务器 S_i、执行任务 J_m、实际所需时间 t_{im}:

$$t_{im} = \frac{R_m}{v_i} \quad (i, m = 1,2,3,\cdots,n) \tag{6}$$

若 $t_{im} \leqslant T_m$,则表示服务器 S_i 能够胜任任务 J_m 的工作,即预计服务器 S_i 能在任务 J_m 的期望时间 T_m 内完成该任务,并将图 G 中的顶点 S_i 和顶点 J_m 相连,得到一条边 $e = S_iJ_m$,将 e 加入图 G 的边集合中,即 $E = E \cup e$;若 $t_{im} > T_m$,则表示以服务器 S_i 当前的负载状态无法及时地完成任务,那么图 G 中的顶点 S_i 和顶点 J_m 不相连。上述遍历完成后即可得到服务器 S 与任务 J 的二部图 G,如图 4 所示。对于服务器 S_i,若与之相连的顶点较多,表示当前该服务器负载较低,能胜任较多的任务;若与之相连的顶点较少,表示当前该服务器负载较高,只能胜任部分任务;若没有与之相连的顶点,表示当前该服务器负载较高,暂时不接收任务。

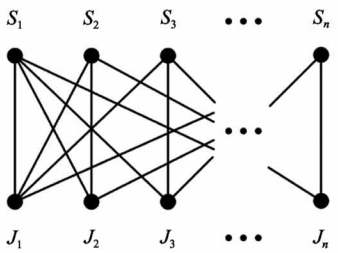


图 4　服务器 S 与任务 J 的二部图 G

管理服务器得到二部图 G 后,根据深度优先的 Edmonds 的匈牙利算法求二部图 G 的最大匹配。遍历顶点集合 S,使顶点 S_i 依次与集合 J 中的顶点进行匹配,若顶点 S_i 与顶点 J_m 相连,且顶点 J_m 还未与集合 S 中的其他顶点匹配,则边 $e = S_iJ_m$ 为一条匹配边,并开始匹配集合 S 的下个顶点;若顶点 S_i 与顶点 J_m 相连,但顶点 J_m 已经与集合 S 中的其他顶点匹配,则寻找其增广路,若找到增广路,则将增广路的匹配边变成未匹配边,而未匹配边变成匹配边,例如找到一条增广路 $S_i - J_1 - S_1 - J_i$,则该增广路匹配边 S_1 J_1 变成未匹配边,未匹配边 S_i J_1 和 S_1 J_i 变成匹配边,并开始匹配集合 S 的下个顶点;若找不到增广路,则顶点 S_i 没有匹配边。集合 S 遍历完后,所有匹配边的集合即为二部图 G 的最大匹配 $M = \{S_iJ_m, \cdots\}$。对于 M 中的匹配边 S_iJ_m,表示以服务器 S_i 当前的负载状况能胜任任务 J_m。

管理服务器得到构建的二部图 G 的最大匹配 M 后,由任务分配模块按照 M 中的匹配边 S_iJ_m,将任务 J_m 发送至服务器 S_i 进行执行。若任务集合 J 中存在没有匹配边的任务,则将该任务移至任务队列队首,重新进行二部图构建和求最大匹配流程。

在整个算法流程中,构建二部图需要对每个服务器与每个任务之间进行判断,因此对于 n 个服务器和 n 个任务,其时间复杂度为 $O(n^2)$。对二部图最大匹配采用深度优先匈牙利算法求解时,二部图一边有 n 个点,最多找到 n 条增广路,若二部图构建完成后有 x 条边,那么每条增广路把所有边遍历一遍,其时间复杂度为 $O(nx)$。而根据二部图构建流程,二部图中最多存在 n^2 条边,因此最坏时间复杂度为 $O(n^3)$,所以整个算法流程的最高时间复杂度为 $O(n^3)$。由于服务器集群的应用场景中服务器个数主要为一位数或两位数,所以管理服务器能很快计算出最大匹配。

重复上述算法流程,管理服务器即可将任务队列中的任务分配至各个服务器中执行,同时低负载的服务器更容易匹配到任务量较大且期望时间较短的任务,高负载的服务器更容易匹配到任务量小且期望时间较长的任务,甚至不会匹配到任务,由此可以实现任务并发时服务器集群中各服务器间的负载均衡。

3　算法仿真与性能分析

本文采用 Java 语言编写管理服务器和服务器集群的程序,系统采用 1 个管理服务器和 4 个服务

器组成的服务器集群,服务器配置参数如表 1 所示。

表 1　服务器配置

服务器名称	服务器配置
管理服务器	Core(TM) i7-7700HQ@2.80 GHz 内存 16 GB
服务器 1	Pentium(R) G645@2.90 GHz 内存 2 GB
服务器 2	双路 Xeon(R) E5-2609@2.40 GHz 内存 16 GB
服务器 3	双路 Xeon(R) E5-2609@2.40 GHz 内存 12 GB
服务器 4	Pentium(R) E5500@2.80 GHz 内存 2 GB

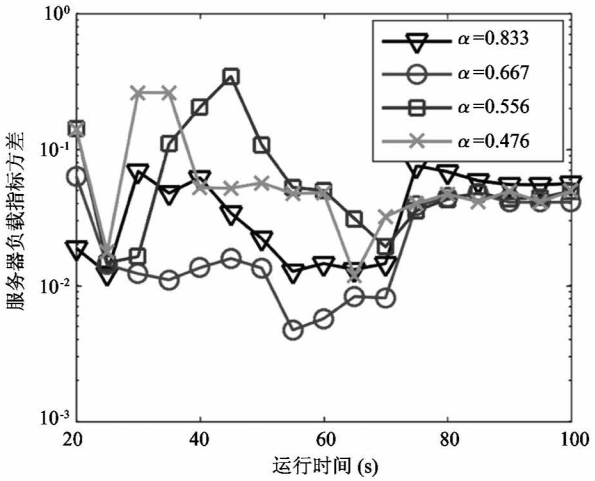
基于本文算法,通过向管理服务器分别发送 500、1 000、1 500 个任务的任务队列,运行 20 s 后每隔 5 s 记录服务器的负载指标,比较不同取值对本文算法的影响,结果如图 5 所示。由图 5 可知,当向管理服务器发送 500 个任务时,α 取 0.667 的负载指标方差大部分时间都低于其他取值的负载指标方差;当向管理服务器发送 1 000 个任务时,α 取 0.667 的负载指标方差大部分时间都低于或接近其他取值的负载指标方差;当向管理服务器发送 1 500 个任务时,20 ~ 50 s 内 4 种取值的负载指标方差都变化较大,50 s 后 α 取 0.667 和 0.833 的大部分时间都低于其他取值的负载指标方差。

服务器分别完成 500、1 000、1 500 个任务所消耗的时间如表 2 所示。由表 2 可知,当 α 取 0.667 时,
服务器完成所有任务消耗时间较少。综合图5和表2,当本文算法的 α 值取 0.667 时,服务器拥有较好的负载均衡度,且完成任务消耗时间较少。

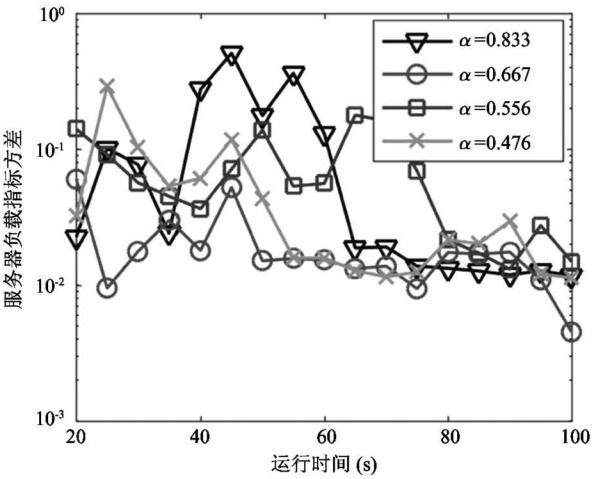
表 2　不同 α 值服务器完成任务消耗时间

α 取值	完成 500 个任务消耗时间(s)	完成 1 000 个任务消耗时间(s)	完成 1 500 个任务消耗时间(s)
0.833	98	185	343
0.667	86	167	254
0.556	137	233	268
0.476	172	241	292

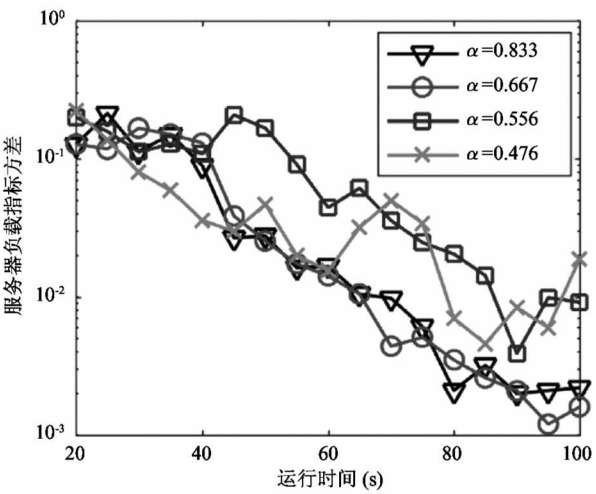
以下实验本文算法的 α 值均取 0.667。通过向管理服务器发送 500、1 000、1 500 个任务的任务队列,对常见的加权轮询调度算法、最小连接数算法和



(a) 500 个任务



(b) 1 000 个任务

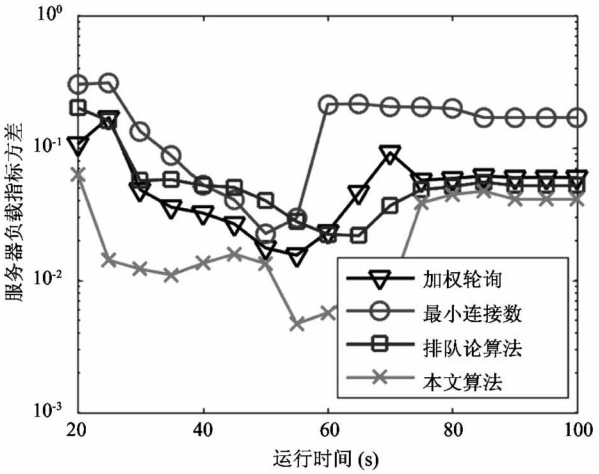


(c) 1 500 个任务

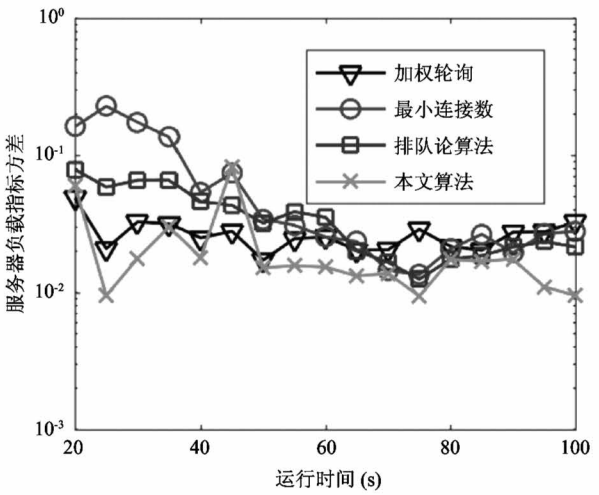
图 5　不同 α 值对本文算法的影响

基于排队论综合指标评估的动态负载均衡算法(以下简称排队论算法)与本文算法进行仿真分析,运

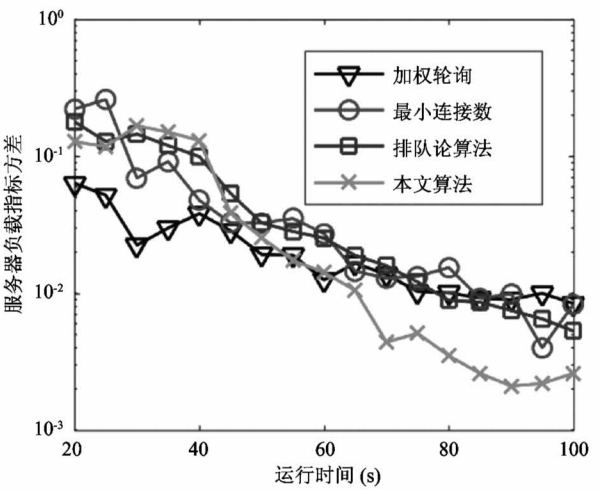
行20 s后每隔5 s记录服务器的负载指标,通过负载指标的方差评估服务器的负载均衡情况,实验结果如图6所示。由图6可知,在3个不同任务队列的情况下,运行20 s后本文算法负载指标的方差大部分时间都低于加权轮询调度算法、最小连接数算法和排队论算法,说明采用本文算法的服务器集群负载均衡度更好。



(a) 500 个任务



(b) 1 000 个任务



(c) 1 500 个任务

图6　不同任务队列的算法效果对比

3种算法完成500、1 000、1 500个任务所消耗的时间如表3所示。根据表3数据,当任务数量较少时,由于加权轮询算法未考虑服务器实时的负载状态,最小连接数算法和本文算法均优于加权轮询算法,但此时本文算法和最小连接数算法差距不大;当任务数量逐渐增大时,本文算法逐渐优于最小连接数算法,消耗的时间更少。

表3　完成队列所有任务消耗的时间

任务数(个)	加权轮询消耗时间(s)	最小连接数消耗时间(s)	排队论算法消耗时间(s)	本文算法消耗时间(s)
500	230	84	87	86
1 000	437	178	172	167
1 500	653	288	269	254

4　结　论

针对服务器集群系统中管理服务器调度任务的场景,本文提出了一种基于二部图最大匹配的动态负载均衡算法,以二部图的匹配结果作为管理服务器的调度方案。仿真实验结果表明,该算法与加权轮询调度算法、最小连接数算法和基于排队论综合指标评估的动态负载均衡算法相比,能使服务器拥有更好的负载均衡度;而且对于相同数量的任务队列,采用该算法的服务器集群能更快地完成所有任务。

由于通过二部图求出的最大匹配不一定是完美匹配,于是就存在取出的任务未匹配到服务器的情况,本文算法将该任务放回任务队列,容易导致该任务的响应时间过长,今后可以在这类任务的处理上对算法进行改进和优化;本文算法求解二部图最大匹配的计算复杂度与服务器数量相关,当服务器规模较大时,算法效率将下降,因此对求二部图最大匹配的匈牙利算法的改进也是本文算法的重要优化方向之一。

参考文献

[1] 王钊. 流媒体服务器集群负载均衡策略的研究[D]. 西安:西安邮电大学计算机学院,2017: 10-16

[2] Wang W, Casale G. Evaluating weighted round robin load balancing for cloud web services[C]//Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 2014: 393-400

[3] Lin C, Chin H, Deng D. Dynamic multiservice load balancing in cloud-based multimedia system[J]. *IEEE Systems Journal*, 2014, 8(1): 225-234

[4] 朱虹宇,李挺,闫健恩,等. 基于动态负载均衡的分布式任务调度算法研究[J]. 高技术通讯,2014,24(12): 1261-1269

[5] Li J, Nie Y, Zhou S. A dynamic load balancing algorithm based on consistent hash[C]//Proceedings of the 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, 2018: 2387-2391

[6] Wen Z, Li G, Yang G. Research and realization of Nginx-based dynamic feedback load balancing algorithm [C]//Proceedings of the 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, 2018: 2541-2546

[7] 王文博,叶庆卫,周宇,等. 基于排队论综合指标评估的动态负载均衡算法[J]. 电信科学,2018,34(7):86-91

[8] 王钊,刘钊远. 一种改进的流媒体集群动态负载均衡调度算法[J]. 计算机与数字工程,2018,46(2):241-246

[9] 杜红艳,周一青,田霖. 一种基于模糊控制的负载均衡决策机制[J]. 高技术通讯,2015,25(10-11):886-894

[10] 朱思峰,沈连丰,柴争义. 基于接入选择和业务转移的异构网络动态负载均衡机制[J]. 高技术通讯,2014,24(10):1007-1013

[11] Ma J, Ding G, Wang R. A new load balancing method based on simulated annealing algorithm in streaming media system[C]//Proceedings of the 8th International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, China, 2012: 1-4

[12] Pan K, Chen J. Load balancing in cloud computing environment based on an improved particle swarm optimization [C]//Proceedings of the 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2015: 595-598

[13] 魏雪. 基于遗传算法的 Web 服务器集群负载均衡的研究[D]. 杭州:浙江理工大学信息学院,2016: 31-43

[14] 谢志远. 关于二部图与匹配问题的研究[D]. 洛阳:河南科技大学数学与统计学院,2014: 21-25

[15] Zhou X, Yang L, Yuan D. Bipartite matching based user grouping for grouped OFDM-IDMA[J]. *IEEE Transactions on Wireless Communications*, 2013, 12(10): 5248-5257

[16] Lu K, Ji R, Tang J, et al. Learning-based bipartite graph matching for view-based 3D model retrieval[J]. *IEEE Transactions on Image Processing*, 2014, 23(10): 4553-4563

[17] 饶磊,汤小春,侯增江. 服务器集群负载均衡策略的研究[J]. 计算机与现代化,2013,15(1):29-32

[18] Edmonds J. Paths, trees, and flowers[J]. *Canadian Journal of Mathematics*, 2012, 17(3): 361-379

Dynamic load balancing algorithm based on maximum matching of bipartite graph

Zhou Lei*, Meng Limin*, Zhou Lipeng*, Jiang Wei**

(*College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023)

(**College of Information Science and Technology, Zhejiang Shuren University, Hangzhou 310015)

Abstract

Load balancing algorithm is one of the core technologies of server cluster, and its key is how to distribute tasks evenly to each server node. Therefore, a dynamic load balancing algorithm based on maximum matching of bipartite graph is proposed. In this algorithm, the ratio of the amount of tasks performed by the server to the actual completion time is taken as the load index of the server, and the load index is feedback to the management server in the cluster system in real time. For tasks to be allocated, a bipartite graph of servers and tasks is constructed, according to the quantity and expected completion time of tasks and the load index of each server. Then the Hungarian algorithm of Edmonds is used to find the maximum matching of the bipartite graph, and the task is sent to the corresponding server according to the matching result in the end. The experimental results show that this algorithm has better load balancing effect and can complete all tasks faster.

Key words: dynamic load balancing, task allocation, sever cluster, bipartite graph, maximum matching