doi:10.3772/j.issn.1002-0470.2014.12.007

基于动态负载均衡的分布式任务调度算法研究①

朱虹宇②* 李 挺** 闫健恩③* 张兆心*

(*哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

(**国家计算机网络应急技术处理协调中心 北京 100029)

摘 要 为提高分布式平台的性能,研究了其任务调度方法。针对分布式任务调度容易引起节点间负载不均衡,从而造成系统资源浪费的问题,提出了一种基于动态负载均衡的分布式任务调度算法。该算法根据各节点的实时性能指标(包括 CPU 利用率、内存使用率、平均负载指标、网络状况)进行任务动态调度,使各节点的负载相对均衡。在分布式平台下进行了拓扑探测、DNS 分布与配置探测实验并对不同算法的性能做了对比。实验结果表明,采用基于动态负载均衡的任务调度算法任务完成时间比轮询调度算法平均减少 30%,比 Min-Min 调度算法平均减少 17%。

关键词 分布式平台,任务调度,负载均衡,任务重组

0 引 言

随着互联网信息量的急剧膨胀和信息种类的大幅增多,进行网络信息探测的难度日益增大,若采用传统的集中式系统,将会花费大量时间,从而影响数据的时效性、可用性,所以研究更加灵活的高性能、易扩展[1]的分布式网络探测方案十分重要。分布式平台的任务调度策略直接影响到分布式平台的性能。任务调度的目标是合理地将待执行任务按一定的优化策略分发给分布式平台的处理单元并行执行,以降低任务的处理时间和提高系统的处理能力及性能[2],因此任务调度策略是分布式平台的关键环节,若任务调度策略不够优化,就会造成分布式平台的总体性能较差。

分布式平台的任务主要分为两类:相互独立的任务和相互依赖的任务[3]。相互依赖的任务主要用有向图进行分析,一般采用静态调度[4]。新泽西理工学院的 Palis 等人[5]和浦项工科大学的 Park 等人[6]从图的角度分析了相互依赖任务的调度问题,提出了基于簇的方法、基于复制的算法,但它们不适用于独立的任务调度。Yagoubi 等人[7]提出在分布式平台中减少任务调度时的消息交互,以减少资源的消耗。Chauhan 等人[8]提出了基于 Min-Min 算法[9]的改进算法,以提高分布式平台服务质量。上述研究主要以分布式平台中的任务为重点,对平台中的节点性能差异没有过多考虑,因此本文重点研究分布式平台中的节点性能对任务执行效率的影响。由于组成分布式平台中的节点是普通的 PC 机,每个客户端节点处理任务时由于即时内存使用

① 国家科技支撑计划(2012BAH45B01),国家自然科学基金(61100189,61370215,61370211),国家信息安全 242 计划(2014A085)和山东省中青年科学家奖励基金(BS2011DX001)资助项目。

② 男,1989 年生,硕士;研究方向:网络安全;E-mail:vipzhy@163.com

③ 通信作者,E-mail:yje@hitwh.edu.cn

(收稿日期:2014-05-08)

率、CPU 利用率、网络性能和任务数量不同,造成各节点的处理性能也有所不同。对网络探测任务来说,网络性能尤为关键。若网络拥塞严重,将导致网络探测任务得不到正确的结果。所以本文提出了一种基于动态负载均衡的分布式任务调度算法(下称基于动态负载均衡的算法),该算法根据节点的网络性能、CPU 利用率、内存使用率和执行任务的数量决定节点任务的调度和下发任务的数量,因而能够提高每个节点任务执行的效率。

1 相关算法

任务调度策略在分布式系统中至关重要。常用的任务调度算法主要包括轮询调度算法[10]、Min-Min 算法等。但这些算法在实际应用中分布式网络探测的效率不高,经常出现某个节点负载过高而有些节点没有任务执行的情况。

1.1 轮询调度算法

轮询调度算法比较简单,它不需要过多的计算和分析,也不需要考虑客户端节点的状态,而是直接将待执行的任务按照任务流顺序依次下发给各节点,空闲节点接收到任务后开始执行,否则放入节点的任务队列中,等待节点空闲时调度。该算法的优点是实现简单,开发和维护成本低,由于服务器端无需考虑节点状态,所以服务器的开销很小,任务完成的时间主要由各个节点性能决定,在任务量不多的情况下系统效率较好。但缺点是没有考虑节点的状态和任务的大小,任务数量较多的情况下,容易造成某些节点负载过重,而有些节点没有任务执行,导致任务的平均完成时间较长,降低了分布式平台的性能和效率。该算法的时间复杂度是 $O(n)$,空间复杂度也是 $O(n)$。

1.2 Min-Min 算法

Min-Min 算法也是分布式平台的常用算法。该算法的核心思想是将预期完成时间最早的任务映射到性能最佳的机器上。该算法需要计算每个任务的期望完成时间,然后找到最早完成时间的任务及对应节点,将该任务分配给对应节点执行。Min-Min 算法考虑了任务完成时间的期望,并将任务下发给性能最好的机器,理论上任务完成效率较高。但在分布式网络探测环境中,待探测的任务复杂程度无法确定,计算出任务完成时间的期望也不准确,同时某些时间期望较长的任务是重要任务,却因先调度时间期望较短的任务而延迟调度。由于该算法将任务优先分配给性能良好的节点,所以容易造成某个节点负载过重[11],而有些节点一直处于空闲状态。该算法的时间复杂度是 $O(n^2)$,空间复杂度是 $O(n)$。

轮询调度算法和 Min-Min 算法在实际应用中处理简单的任务时效率较好,但在处理复杂的任务时都会导致节点间负载不均衡的情况,降低分布式平台的效率。

2 基于动态负载均衡的算法

在分布式网络探测平台中,由于探测任务跟网络有关,和传统的计算任务相比更为复杂,受节点的资源制约,所以需要用更加合理的算法提高分布式探测平台的效率。

本文提出了一种基于客户端的负载情况,动态决策待调度任务的数量和动态选择客户端节点的算法,该算法运行在 Linux 系统,根据客户端的性能指标(包括网络性能、CPU 和内存利用率、平均负载参数)调度和分配任务数,比静态调度更加灵活、更具合理性。

在分布式平台中,每个客户端节点都是异构的,同一时刻每个节点的性能是不同的。因此根据各节点的性能情况分配不同数量的任务,使得分布式中各节点负载较为均衡,效率也会提升。而对于网络探测任务来说,除了节点的 CPU 利用率、内存使用率、内存的大小和已有的任务数量对执行效率有影响之外,最重要的是网络状况这一衡量指标。本算

法主要由两部分组成,分别是任务动态重组和任务下发,如图 1 所示。当有任务需要执行时,先通过任务重组将任务分解成更小的子任务,然后任务下发模块分析节点性能,选择可执行任务的节点,并将子任务下发给节点。

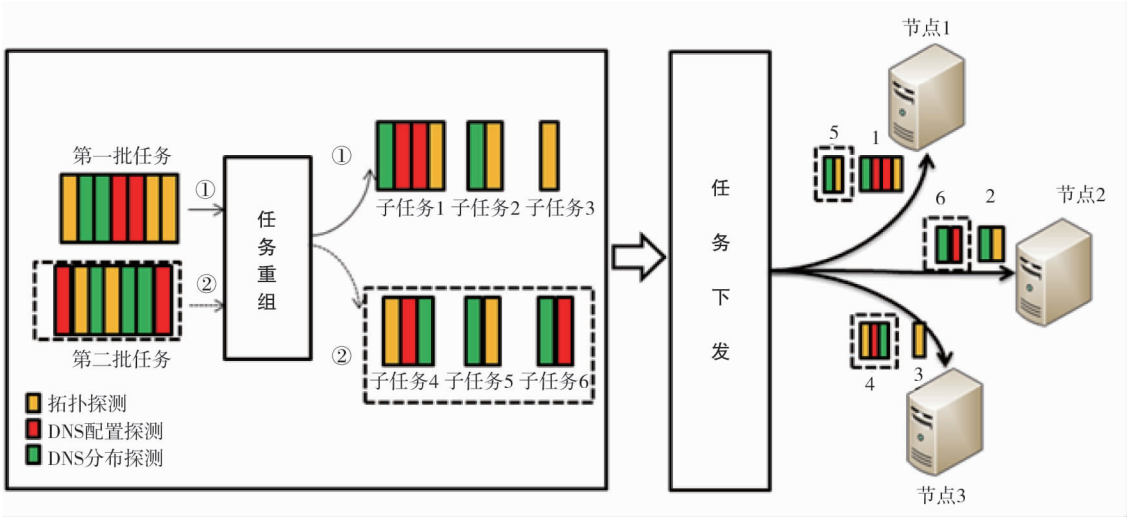


图 1　分布式调度流程

2.1　可调度节点的选取方法

任务下发时,考虑节点已分配的任务量、平均负载参数以及客户端节点到服务器的网络 PING 值,并根据这些值进行动态调度。任务量表明了节点上未完成的任务数目,如果节点有过多任务没有完成,将不会给该节点分配任务,否则会出现任务集中在某些节点上的情形;平均负载参数(load average)是 Linux 系统中一个自带的参数,表示在特定时间间隔内运行队列中的平均进程数,当前系统的平均负载值可通过 top 命令查看,该参数是描述 Linux 系统性能的关键指标,所以只给平均负载较低的主机分配任务,否则分配了任务也无法尽快完成,影响平台效率;网络 PING 值表明了客户端节点与服务器之间的网络状况,由于节点与服务器之间需要传送大量的数据,同时客户端的探测任务也需要良好的网络,网络状况不好时,会存在大量的丢包现象,探测效率也会降低,所以,网络状况不理想时,也不会继续分配任务,只有在上述性能指标都满足的前提下,该节点才是合适的节点,并考虑为其分配任务。本文提出公式

$$\begin{cases} Q = M'k_1 + Lk_2 + Nk_3 \\ M' = \log(M) \end{cases} \tag{1}$$

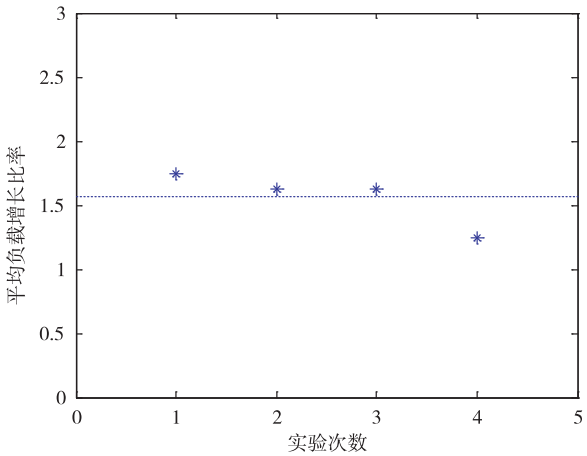
定义 Q 表示计算客户端节点的综合情况,并定义可调度任务的阈值 Q_t,只有 Q 值小于定义的阈值 Q_t时,才为节点调度任务。式中,M 为节点待执行任务数量,M'为 M 归一化后的值,L 为操作系统的平均负载参数,N 为 PING 值的响应时间。k_1、k_2和 k_3是描述各参数的权重,根据不同的任务,可以动态调整相应的 k 值。因为任务数量的量级比平均负载和网络 PING 值大,所以使用对数函数对 M 进行归一化处理,使三个参数在同一个量级上。当计算出的 Q 值小于设定的阈值 Q_t时,则该节点满足调度条件。若有多个节点满足条件,则选取 Q 值最小的节点进行调度。

因为不同的分布式任务 k_1、k_2和 k_3的取值不同,所以需要通过实验确定权值的取值。确定权值的方法是通过多次增加下发任务的数量,然后计算任务数量、平均负载和网络参数的平均增长率的比值,该比值反映出了各权值的重要程度,所以 k_1、k_2和 k_3取值为相应参数的比值。以域名系统(DNS)配置探测

为例,表 1 给出了分布式平台进行 DNS 配置探测时的相关信息,根据这些信息计算出的平均负载参数和网络参数的增长率如图 2 所示。当下发的任务数量增长比率为 2 时,可以从图中看出平均负载的增长较为缓慢,增长比率在[1.25,1.75]区间,网络 PING 值增加趋势最为显著,增长比率在[2.43,3.13]区间,三者的增长比例约为 2∶1.56∶2.75,所以分布式网络探测平台中,网络性能相对于其他指标更为重要。因此该任务的 k_1 取值为 2,k_2 取值为 1.6,k_3 取值为 2.8 较为合理。若 k_1、k_2 和 k_3 的取值都为 2 时,与 k_1 取值 2、k_2 取值 1.6、k_3 取值 2.8 相比,在实际执行任务时前者完成任务所需要的时间是后者的 1.85 倍。在执行其他任务时,可通过多次实验评估确定 k_1、k_2 和 k_3 的取值。

表 1　DNS 配置探测测试结果

任务数量	平均负载	网络 PING 值(s)	完成时间(s)
125	0.12	0.007	34
250	0.21	0.017	97
500	0.34	0.045	225
1000	0.55	0.141	623
2000	0.69	0.395	1055



(a)负载参数增长比率　　(b)网络参数增长比率

图 2　性能参数增长比率

2.2　任务动态重组

通过上述过程选取可执行任务的节点后,第二步是进行任务重组。任务重组就是根据节点的性能,动态地生成任务项目数量。通过实验发现,节点处理能力与自身的内存使用率和 CPU 利用率有关,而节点处理能力又决定了执行任务的数量,所以下发的任务数量不是固定值。任务项目数是根据下式计算而得:

$$\begin{cases} N = (1-C)\alpha + (1-M)\beta \\ 0 < C \leqslant 1 \\ 0 < M \leqslant 1 \end{cases} \tag{2}$$

其中,N 表示下发的任务数量,C 表示 CPU 利用率,M 表示内存使用率,α 表示 CPU 权值,β 表示内存权值。依据此公式计算任务项数 N 时,N 始终小于 $\alpha+\beta$,并且受 CPU 使用率和内存使用率的影响而不固定。极限情况下,当 CPU 利用率和内存使用率都很高时,N 值为 0。采用该方法,能很好地控制任务项数的上限 $\alpha+\beta$,并且可根据性能实现任务项数的动态计算。但在实际情况中,还需要考虑任务调度的时间和计算节点性能的时间。假设 T_N 表示完成 N 个任务需要的时间,T_A 表示中心服务器计算节点性能需要的时间,T_S 表示调度任务所需的时间,则 T_N、T_A、T_S 需要满足公式

$$T_N >> T_A + T_S \tag{3}$$

的条件。即任务的执行时间远大于计算节点性能的时间与调度任务所需的时间之和,否则会造成计算过程和调度过程中消耗大量的时间,而执行任务的过程消耗很少的时间,造成了严重的资源浪费,降低了分布式平台的效率。因此,需要根据任务的实际情况设定 N 的下限值。

在上述研究基础上建立基于动态负载均衡的分布式任务调度算法如下:

输入:任务队列 JQ、节点集合 $Nodes$、权值 k_1、k_2、k_3、α、β

BEGIN

$k1,k2,k3,\alpha,\beta$ 初始化;

while 任务队列 JQ 非空 do

for 节点 c in Nodes do

利用公式(1)计算 $Q_c \leftarrow M_c k_1 + L_c k_2 + N_c k_3$;

if $Q_c <= Q_t$ do

将该节点加入到可调度节点队列 NQ;

end for

从 NQ 中选取 Q 值最小的节点 n ;

利用式(2)计算任务数量 $N \leftarrow (1 - C_n)\alpha + (1 - M_n)\beta$;

从任务队列 JQ 中选取 N 个任务;

给节点 n 下发组合后的任务;

删除已调度任务;

end while

END

该算法的时间复杂度是 $O(n^2)$,空间复杂度是 $O(n)$。通过上述算法,减少了客户端节点负载不均衡情况的发生,大部分节点能保持良好的性能,从而提高分布式网络探测平台的处理效率。

3 实验结果

在分布式探测平台的环境中,部署了 4 台 PC 机,其中 1 台为中心服务器,负责收集节点信息和进行任务调度,其余 3 台为执行任务节点,拓扑结构如图 3 所示。

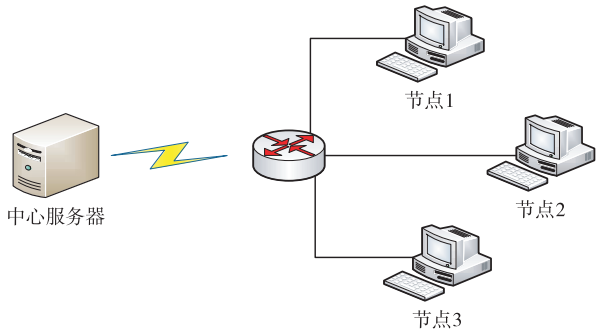


图 3 测试环境拓扑结构

分布式网络探测平台的效率和可调度阈值 Q_t 有关。当 Q_t 定义过高时,导致有些空闲节点无法得到调度;当 Q_t 定义过低时,会导致节点收到过多任务,导致节点间负载不均衡。所以应根据平台的具体任务和运行情况确定 Q_t 的值。为此,可以在一定任务量的情况下进行多次实验,分析完成所有任务花费的时间和 Q_t 的曲线关系,找到完成任务所花费时间最少的点对应的 Q_t 值,该值即为可调度任务的最小阈值。图 4 为完成 1000 项探测任务所需时间和 Q_t 的曲线图,从图中可以看出 Q_t 取值为 2.5 较为合适。

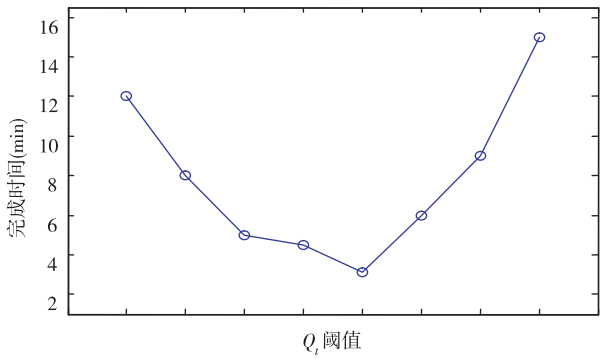


图 4 阈值与任务完成时间的关系

为验证算法的效率,与轮询调度算法和 Min-Min 算法进行了对比实验。规定待探测任务的数量

为 3000 条,其中拓扑探测任务、DNS 分布探测任务、DNS 配置探测任务各 1000 条。公式中 α 和 β 的权值均为 10,理想情况下每次调度的最大任务数量为 20 条。轮询调度算法每次分配任务为 20 条。而在采用 Min-Min 算法时需要计算每个任务完成时间的期望,对于网络探测来说无法算出具体的时间期望,因为任务的完成时间由自身的复杂情况和客户端所在的地域决定。根据多次测试的结果可以总结出 DNS 分布探测所需的时间最短,其次是 DNS 配置探测,花费时间最长的是拓扑探测任务,花费时间的比例大约为 1∶1.3∶1.8。而客户端节点的性能状态通过心跳包每 5 秒向服务器汇报一次,服务器在调度时动态计算节点的负载情况。

每种类型任务采用轮询调度算法、Min-Min 算法和本文提出的动态负载均衡的调度算法进行测试,任务完成时间的测试结果如图 5 所示。采用三种任务调度算法时,节点的 CPU 利用率、内存使用率、系统平均负载和网络负载的平均值对比如图 6 所示。

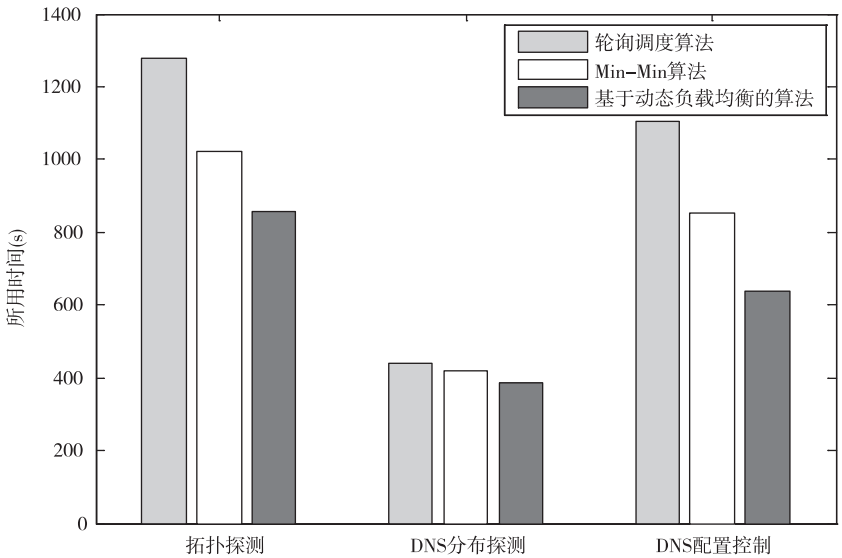
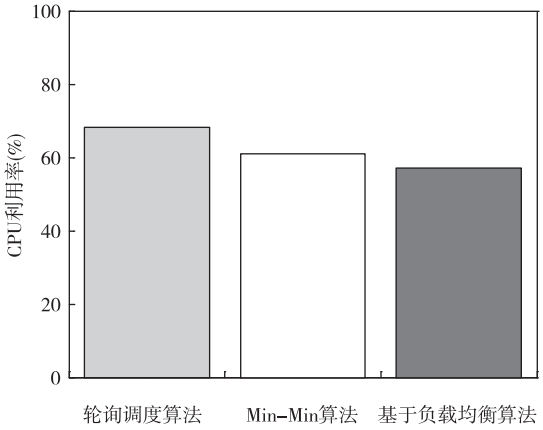


图 5 调度算法对比实验结果

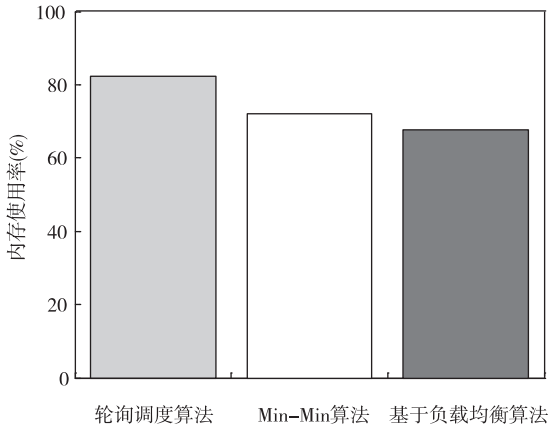
从图 5 中可以看出,采用本文算法后拓扑探测任务的完成时间比采用轮询算法减少 33%,比采用 Min-Min 算法减少 16.2%;DNS 分布探测任务的完成时间比采用轮询算法减少 12.7%,比采用 Min-Min 算法减少 8.3%;DNS 配置探测任务的完成时间比采用轮询算法减少 42.3%,比 Min-Min 算法减少 25.4%。综合比较,采用本文算法完成任务所需时间较短,与轮询调度算法相比任务完成时间平均减少 30%,与 Min-Min 调度算法相比任务完成时间平均减少 17%。从图 6 中的对比数据可以看出,在采用负载均衡调度算法时,CPU 利用率、内存使用率、系统平均负载和网络性能都比采用轮训调度算法和 Min-Min 算法好,消耗的系统资源更少。

上述实验采用基于动态负载均衡的调度算法执行任务时,3 台节点的 CPU 利用率、内存使用率、系统平均负载和网络性能参数如图 7 所示。

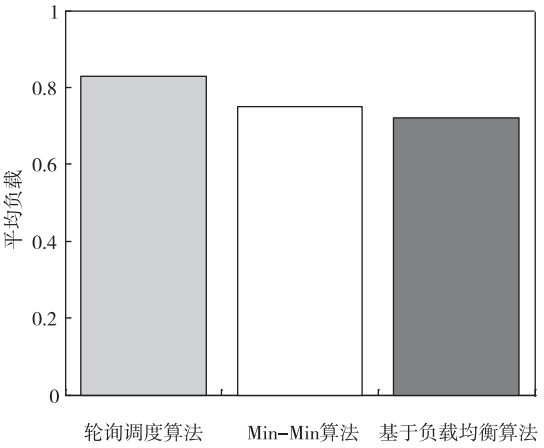
从图 7 可以看出,3 台节点的 CPU 利用率分布在[56.4%,58.2%]区间,内存使用率分布在[66.2%,69.2%]区间,系统平均负载分布在[0.68,0.77]区间,网络 PING 值分布在[109,125]区间。上述实验结果表明,分布式网络探测平台中各节点的 CPU 利用率、内存使用率、系统平均负载和网络性能参数相对均衡、稳定,从而提高了分布式平台的效率,减少了任务的执行时间。



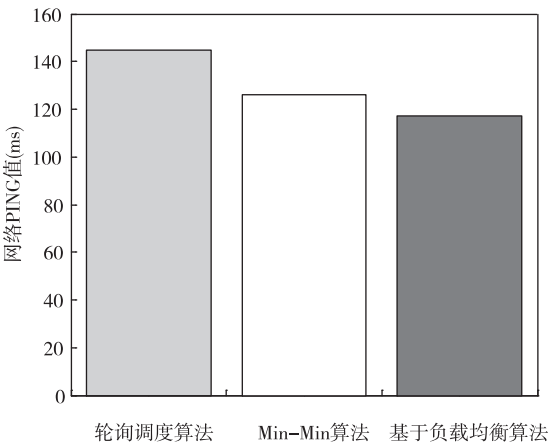
(a) CPU利用率对比



(b) 内存使用率对比

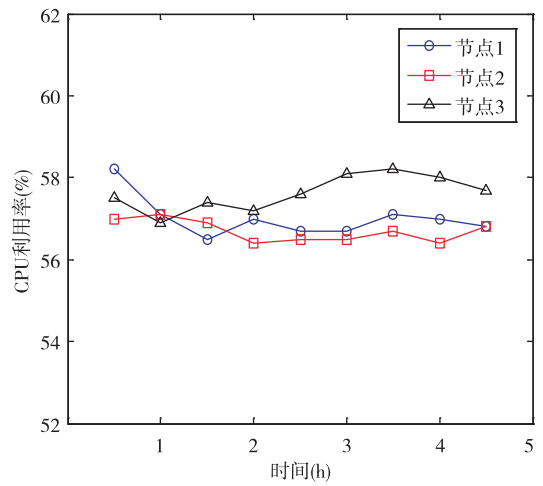


(c) 系统平均负载对比

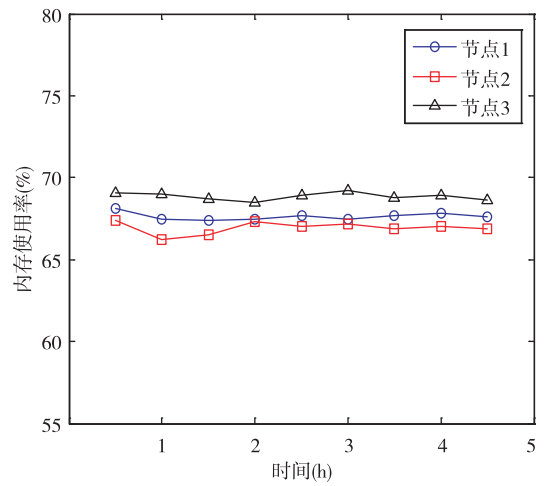


(d) 网络性能对比

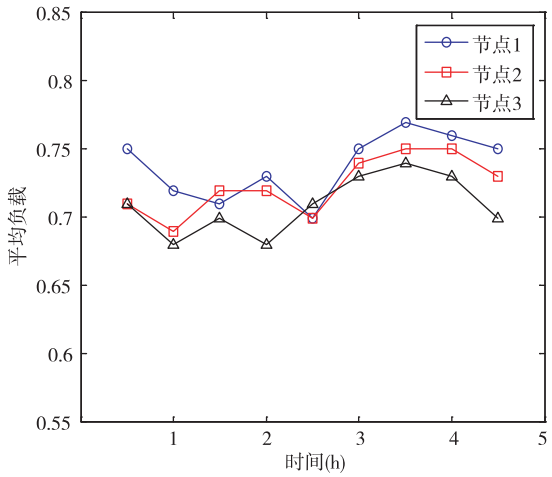
图 6　调度算法性能参数对比



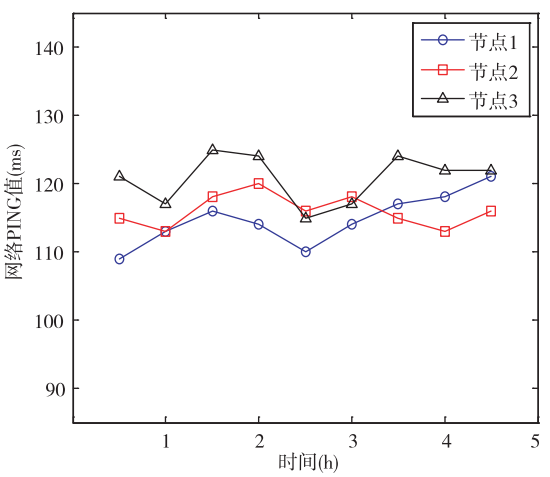
(a) CPU利用率



(b) 内存利用率



(c) 系统平均负载



(d) 网络性能

图 7 分布式节点性能参数测试

4 结 论

在分布式网络探测平台中,传统的任务调度算法无法满足要求,容易造成节点负载不均衡,且效率低。本文提出基于动态负载均衡的任务调度算法,根据获取每个客户端节点的性能参数,通过公式进行任务重组并选出调度节点。以分布式网络探测平台为例,任务调度算法分别采用轮询调度、Min-Min 算法和动态负载均衡调度算法,对比结果显示采用基于动态负载均衡的算法与轮询调度算法相比,任务完成时间平均减少 30%,与 Min-Min 算法相比任务完成时间平均减少 17%。实验结果表明,基于动态负载均衡调度算法在使各节点的 CPU 利用率、内存使用率、系统平均负载和网络 PING 值相对均衡的同时,提高了分布式平台的任务执行效率。

参考文献

[1] Stankovic J A. Stability and distributed scheduling algorithms. *IEEE Transactions on Software Engineering*, 1985, 11(10): 1141-1152

[2] Saxena S, Khan M Z, Singh R. Performance analysis in distributed system of dynamic load balancing using fuzzy logic. In: Proceedings of the IEEE 2012 Spring Corgress on Engineering and Technology, Xi'an, China, 2012. 1-5

[3] Amudha T, Dhivyaprabha TT. QoS priority based scheduling algorithm and proposed framework for task scheduling in a grid environment. In: Proceedings of 2011 IEEE International Conference on the Recent Trends in Information Technology, Chennai, India, 2011. 650-655

[4] 何琨,赵勇,陈阳. 分布式环境下多任务调度问题的分析与求解. 系统工程理论与实践,2007,27(5):119-125

[5] Palis M A, Liou J C, Wei DSL. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 1996, 7(1): 46-55

[6] Park C I, Choe T Y. An optimal scheduling algorithm based on task duplication. *IEEE Transactions on Computers*, 2002, 51(4): 444-448

[7] Yagoubi B, Slimani Y. Task load balancing strategy for grid computing. *Journal of Computer Science*, 2007, 3(3): 186-194

[8] Chauhan S S, Joshi R C. QoS guided heuristic algorithms for grid task scheduling. *International Journal of Computer Applications*, 2010, 2(9): 24-31

[9] Kokilavani T, Amalarethinam D I. Load balanced Min-Min algorithm for static meta-task scheduling in grid computing. *International Journal of Computer Applications*, 2011,

20:43-49

[10] 王友亮,叶柏龙. 分布式系统中动态负载平衡的研究. 科学技术与工程,2005,5(9):572-575

[11] Huankai C, Wang F, Helian N. User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In: Proceedings of the 2013 National Conference on Parallel Computing Technologies (PARCOMPTECH), Bangalore, India, 2013. 1-8

Research on a distributed task scheduling algorithm based on dynamic load balancing

Zhu Hongyu*, Li Ting**, Yan Jianen*, Zhang Zhaoxin*

(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

(**National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, 100029)

Abstract

To improve the performance of distributed systems, the distributed task scheduling was studied. Considering that distributed task scheduling easily causes load imbalance between nodes, so causing the wasting of system resources, a distributed task scheduling algorithm based on dynamic load balancing was proposed. The algorithm conducts the dynamic task scheduling according to the real-time performance indicators of CPU utilization, memory utilization, average load and network performance of each node to make the load of each node relatively balanced. The experiments on topology detection, DNS distribution detection and DNS configuration detection were performed under a distributed system to compare the performance of the proposed algorithm with other two algorithms. The results showed that the task completion time of the proposed algorithm was reduced by 30% compared to the polling scheduling algorithm, and was reduced by 17% compared to the Min-Min scheduling algorithm.

Key words: distributed system, task scheduling, load balancing, task reorganization