doi: 10.12068/j.issn.1005-3026.2019.06.010

一种适应性的动态负载平衡模型

赵廷磊, 乔建忠, 林树宽, 王彦华
(东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

摘 要: 为了改善分布式系统中负载分布不平衡对性能的影响,提出并实现了一个基于控制理论的时滞脉冲切换负载平衡模型. 该模型根据节点资源的动态性建立了相关子系统. 当节点状态发生改变时触发子系统的切换,并根据负载迁移规则对过量负载进行迁移,迁移比例根据节点的实时运行状态进行计算. 节点仅在此时才进行信息广播,降低了通信开销,提升了动态负载平衡的效率. 给出了相应的负载平衡算法,并在实际平台上进行了验证. 实验结果表明,与其他负载平衡算法相比,本模型算法使负载平衡时间平均减少29.82%.

关 键 词: 负载迁移;节点状态;脉冲切换系统;动态负载平衡;分布式系统
中图分类号: TP 391 **文献标志码**: A **文章编号**: 1005-3026(2019)06-0813-06

An Adaptive Dynamic Load Balancing Model

ZHAO Ting-lei, *QIAO Jian-zhong*, *LIN Shu-kuan*, *WANG Yan-hua*
(School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: ZHAO Ting-lei, E-mail: zhaotinglei_0928@163.com)

Abstract: To improve the distributed system performance which is affected by load unbalance, this paper proposes and implements an impulsive and switching load balancing model with time delay based on control theory. The model constructs the corresponding sub-system according to the dynamics of node resources. The overloading is migrated by the load migrate rule, of which the migrate proportion is calculated by real-time running states of nodes. Only on this moment, the node broadcasts its message to others. Thus, the communication cost among nodes decreases, and the efficiency of dynamic load balancing increases. This paper provides the corresponding load balancing algorithm and conducts the evaluation on a real platform. Experimental results demonstrate that compared with other load balancing algorithms, the load balancing time of the proposed model is reduced by 29.82% on average.

Key words: load migration; node state; impulsive and switching system; dynamic load balancing; distributed system

负载平衡是分布式系统的一个重要研究领域,节点间负载的不平衡会严重影响工作效率. 在分布式系统中,每个节点可多次提交不同类型的新任务并动态地加入和退出[1]. 处理大规模任务时,将任务划分为多个子任务并分配到各个节点处理,返回的结果由系统重新融合. 若某些节点繁忙不能及时返回结果,将严重影响系统性能. 现有的负载平衡研究分为以下几个方面:1)通过使用典型的控制模型进行负载平衡[2]. 分布式控制模型的主要特征是节点之间可以自主地相互协调以进行任务分配. 通过模型控制可以适应不同的系统规模,并允许一些节点出现故障,因此鲁棒性很高. 但由于每个节点只能知道相邻节点的信息,结果可能只是局部最优的. 节点之间的协商会产生额外的计算成本. 2)负载平衡中的一些典型的资源优化方法. 其中,自主资源优化方法[3]只考虑节点本身的资源,忽略节点之间资源的协调,该方法仅在大多数节点可以自主完成分配的任务的情

收稿日期: 2017-10-27
基金项目: 国家自然科学基金资助项目(61402442); 辽宁省软件系统开发与应用重点实验室项目.
作者简介: 赵廷磊(1983-),男,甘肃白银人,东北大学博士研究生; 乔建忠(1964-),男,辽宁兴城人,东北大学教授,博士生导师.

况下才能很好地执行. 基于资源的上下文优化方法[4]考虑了节点之间的协调,适用于协作分布式系统,但是这些优化方法需要大量的计算,在大规模系统中会产生高成本. 3)实现负载平衡可靠性有基于冗余的方法[5]和基于非冗余的方法[6]. 针对某些特殊情况,使用启发式方法以最低的成本实现理论上可靠的性能. 4)利用典型协调机制对异构节点之间进行协调以便实现负载平衡. 异构节点之间的协调机制常用博弈论[7]和图论[8]. 节点的角色总是假定在任务分配和执行期间保持固定. 节点可以动态地改变在系统中的角色,根据自己的策略进行单独决策. 5)根据网络结构的特征在不同的网络拓扑结构中实现负载平衡. 网络结构影响了节点之间的协调以及执行任务. 通常有两种网络结构:节点通过物理通信网络结构互连[9],在任务分配中应考虑节点的资源和位置;节点通过虚拟交互网络结构互连[10].

本文结合脉冲切换系统模型和时滞系统模型的特点针对分布式动态负载平衡过程提出了一个时滞脉冲切换负载平衡模型. 该模型是由 4 个连续的子系统构成的一个连续 - 离散系统. 在进行负载迁移时,根据本节点和其他所有节点的处理能力计算迁移比例,只有空载和轻载节点接收迁移的负载,同时兼顾了各状态的节点. 节点状态改变时通过脉冲信号切换系统,并在此时广播节点信息,降低通信开销,提高负载平衡的效率.

1 时滞脉冲切换负载平衡模型

本系统包含多个连续的子系统,子系统之间是离散的,属于连续 - 离散系统. 每个节点在不同时刻根据自身状态对应不同的子系统,各个子系统在状态改变时触发切换. 不同的子系统交替发生、相互作用,最终使系统达到平衡状态,并使性能得到改善.

1.1 时滞脉冲切换负载平衡过程描述

分布式系统中一个典型的时滞脉冲切换负载均衡过程可由图 1 来描述. 节点 2 为重载节点,它将过量负载迁移到轻载节点 3 和空载节点 4 上执行. 最终各节点均处于适载状态,系统达到平衡.

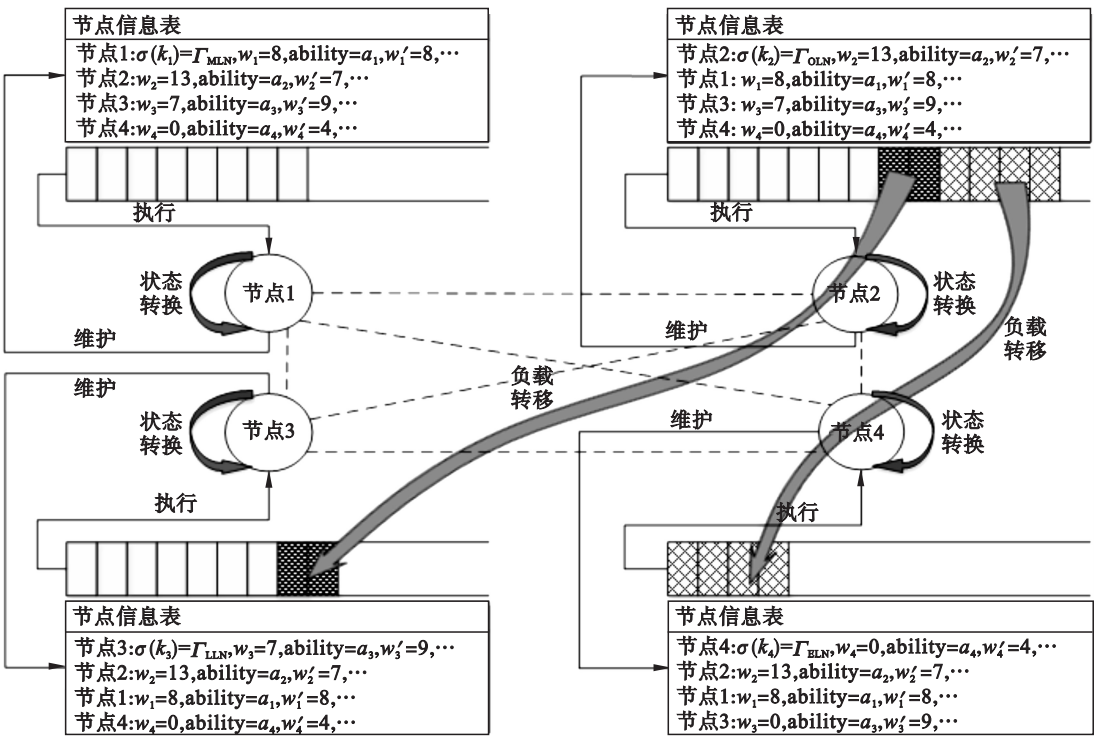

图 1 4 节点分布式系统的时滞脉冲切换负载均衡过程

Fig. 1 Process of time delay impulsive switching load balancing model for distributed system with 4 nodes

1.2 时滞脉冲切换负载平衡模型

系统由 n 个节点 $P_i(0\leqslant i\leqslant n-1)$ 组成. 定义 P_i 的处理能力 a_i 是 P_i 的核心数 Nc_i, 频率 F_i,内存 M_i 和当前利用率 $U_i(t)$ 的综合指标:

$$a_i = \begin{cases} \Xi_i, & 0 \leqslant U_i(t) \leqslant \alpha; \\ \Xi_i \times \dfrac{1-(\beta-\alpha)^2}{\ln(2-\beta)} \ln(2-U_i(t)), & \beta < U_i(t) \leqslant 1; \\ \Xi_i \times (1-(U_i(t)-\alpha)^2), & \alpha < U_i(t) \leqslant \beta. \end{cases} \tag{1}$$

其中：$\Xi_i = \mathrm{Nc}_i(F_i + \varepsilon M_i)$，$\varepsilon = 0.2$；$\alpha = 0.2$；$\beta = 0.55$.

定义 1（节点状态判断依据）：当节点 P_i 的负载分别满足条件 $w_i < w^{\inf}$，$w^{\inf} < w_i < w^{\sup}$，$w_i > w^{\sup}$ 或 $w_i = 0$ 时，P_i 为轻载、适载、重载或空载节点. 其中，$w^{\sup} = w_i'(1+\xi)$，$w^{\inf} = w_i'(1-\xi)$，$\xi = 0.05$.

节点 P_i 的负载为 $w_i(0 \leqslant i \leqslant n-1)$，在子系统的第 k 个驻留时间段 $[t_k, t_{k+1})$ 内所能处理的负载为理想负载 w_i'. 系统启动时，节点 P_i 将其状态和负载信息进行广播，并收集其他节点的信息. 节点 P_i 使用式(2)计算系统当前状态.

$$\text{state} = \begin{cases} 1, & \omega > t; \\ 0, & \omega \leqslant t. \end{cases} \quad \omega = \frac{\sum_1^n w_i}{\sum_1^n a_i}. \tag{2}$$

当 state = 1 时，系统超载，使用 $w_i' = \omega w_i'$ 对 w_i' 进行处理，则 state 转换为 0. 当 state = 0 时，使用时滞脉冲切换负载均衡模型(3)处理.

$$\left.\begin{aligned} &\dot{w}_i(t) = f_{\sigma_i(k)}(t, w_i(t), w_i(t-\tau_i)), && t \in [t_k, t_{k+1}), \sigma_i(k) \in \{\Gamma_{\mathrm{OLN}}, \Gamma_{\mathrm{MLN}}, \Gamma_{\mathrm{LLN}}, \Gamma_{\mathrm{ELN}}\}; \\ &\Delta w_i(t_{k+1}) = w_i(t_{k+1}) - w_i(t_{k+1}^-), && k = 0, 1, 2, \cdots; \\ &w_i(\theta) = w_i(t_0), && \theta \in [t_0 - \tau_i, t_0]. \end{aligned}\right\} \tag{3}$$

其中：$w_i(t) \in \mathbf{R}^n$ 是系统的状态变量，表示节点 P_i 的负载队列长度；$\sigma_i(k)$ 为节点 P_i 在第 k 个时间段 $[t_k, t_{k+1})$ 内被激活的子系统，$t_{k+1} - t_k$ 是子系统的驻留时间；Γ_{OLN}，Γ_{MLN}，Γ_{LLN} 和 Γ_{ELN} 表示重载、适载、轻载和空载状态分别对应的子系统 σ_1，σ_2，σ_3 和 σ_4；$\sigma_l(l = 1, 2, \cdots, 4)$ 在 $[t_0, t]$ 内驻留的总时间为 T_l；切换时刻序列满足 $0 \leqslant t_0 < t_1 < \cdots t_k < t_{k+1} < \cdots$；系统时滞 $\tau_i > 0$；f 是一个分段连续向量值函数，$f(t,0,0) \equiv 0$，$t \in \mathbf{R}^+$；$\Delta w_i(t_{k+1})$ 表示状态跃变；初值条件为 $w_i(\theta) = w_i(t_0)$；$w_i: [t_0 - \tau_i, t_0] \to \mathbf{R}^n$ 除了有限个点 t 外是连续的，而在 t 处满足 $w_i(t^+)$ 和 $w_i(t^-)$ 存在并相等；$w_i(t_0)$ 表示节点 P_i 在初始时刻负载队列的长度.

定义 2（负载迁移规则）：对于处于状态 $\sigma_i(k)$ 和 $\sigma_j(k)$ 的任意两个节点 P_i 和 P_j，负载按 $\{P_i \xrightarrow{u_i(t)} P_j \mid \sigma_i(k) = \Gamma_{\mathrm{OLN}}, \sigma_j(k) \in \{\Gamma_{\mathrm{LLN}}, \Gamma_{\mathrm{ELN}}\}\}$ 进行迁移. 其中 u_i 表示单位时间内从节点 P_i 上迁移到节点 P_j 上的过量负载.

节点的 a_i 不同，每次迁移的负载量各不相同，对节点性能的影响也各不相同. 若按固定划分，会使节点的状态发生频繁的转换. a_i 较小的节点应该比 a_i 较大的节点每次接收的负载量要少，因为其 w_i' 较小，当超过 w_i' 时，系统的状态将发生变化. 为了保证系统 的稳定性，a_i 较大的节点每次迁移出的负载量不宜过大. 因此，使用式(4) 计算适合的负载迁移比例 p_i.

$$p_i = \frac{1}{n}\left(1 - \frac{a_i(t)}{\sum_1^n a_i(t)}\right). \tag{4}$$

根据定义 1 判断节点的状态，当节点 P_i 处于重载状态时，除了处理自身任务和生成新任务外，只将自身过量负载迁移给轻载和空载节点 P_j，而不再接收其他节点迁移的负载. 迁移过程按照负载迁移规则进行，负载迁移比例根据式(4)计算. 针对该过程使用控制理论建立子系统模型，其状态空间具体描述如下：

Γ_{OLN}：

$$\begin{cases} \dfrac{\mathrm{d}w_i(t)}{\mathrm{d}t} = \nu_i(t) - \mu_i(t) - \sum\limits_{i \neq j} p_i u_i(t - \eta_{ij}), \\ y_i(t) = w_i(t) - w_i'(t), \\ u_i(t) = k_i y_i(t). \end{cases} \tag{5}$$

当节点 P_j 处于轻载状态时，除了处理自身任务和生成新任务外，只接收其他重载节点 P_i 迁移的负载，而不再对自身的负载进行迁移. 若接收到了迁移负载，其状态空间具体描述为式(6). 若没有接收到迁移负载，则根据式(7)进行状态转换.

Γ_{LLN}：

$$\begin{cases} \dfrac{\mathrm{d}w_j(t)}{\mathrm{d}t} = \nu_j(t) - \mu_j(t) + \sum\limits_{j \neq i} p_i u_i(t - \eta_{ji}), \\ y_j(t) = w_j(t) - w_j'(t), \\ u_j(t) = 0. \end{cases} \tag{6}$$

当节点 P_i 处于适载状态时，只处理自身任务和生成新任务，既不接收其他节点迁移的负载，也不对自身的负载进行迁移. 其状态空间具体描述为

$$\Gamma_{\mathrm{MLN}}:\begin{cases}\dfrac{\mathrm{d}w_i(t)}{\mathrm{d}t}=\nu_i(t)-\mu_i(t)\ ,\\ y_i(t)=w_i(t)-w_i'(t)\ ,\\ u_i(t)=0\ .\end{cases}\tag{7}$$

当节点 P_j 处于空载状态时,只接收其他重载节点 P_i 迁移的负载,而自身没有可处理的任务,也没有新任务生成,也不迁移负载. 若接收到了迁移负载,其状态空间具体描述为式(8). 若没有接收到迁移负载,节点状态不改变.

$$\Gamma_{\mathrm{ELN}}:\begin{cases}\dfrac{\mathrm{d}w_j(t)}{\mathrm{d}t}=\sum\limits_{j\neq i}p_iu_i(t-\eta_{ji})\ ,\\ y_j(t)=w_j(t)-w_j'(t)\ ,\\ u_j(t)=0\ .\end{cases}\tag{8}$$

式(5)~(8)中:状态变量 w_i 表示节点 P_i 的负载队列长度;输入变量 u_i 表示单位时间内从节点 P_i 上迁移到其他节点上的过量负载;输出变量 y_i 表示节点 P_i 上的过量负载;w_i' 表示节点 P_i 的理想负载队列长度;k_i 表示单位时间内负载平衡系统的闭环增益;p_i 表示节点 P_i 的负载迁移比例;η_{ij} 表示节点 P_i 到节点 P_j 的传输时延;ν_i,μ_i 分别表示节点 P_i 的任务生成率和处理率,使用式(9)和式(10)计算:

$$\nu_i=r_iw_i(t),\tag{9}$$

$$\mu_i=\frac{a_i(t)}{\int_{t_k}^{t_{k+1}}a_i(t)\,\mathrm{d}t}w_i(t).\tag{10}$$

其中,$r_i\in\mathbf{R}^+$,其值由负载的固有属性确定.

2 时滞脉冲切换负载平衡算法

1) 根据式(1)计算系统中节点 P_i 的处理能力 a_i.

2) 依据定义 1 判断节点 P_i 所处的状态,并计算节点 P_i 的理想负载 w_i'.

3) 依据式(2)判断系统当前状态. 当系统超载时,对理想负载 w_i' 进行处理.

4) 根据定义 2 对系统中处于 Γ_{OLN} 状态的重载节点进行负载迁移. 使用式(4)计算适合的负载迁移比例 p_i. 根据式(11)计算单位时间内从节点 P_i 上迁移到节点 P_j 上的过量负载.

$$u_{ij}(t)=p_{ij}(t)(w_i-w_i').\tag{11}$$

5) 根据定义 1 判断节点 P_i 所处的状态,如果系统中还存在重载节点,返回步骤 4). 否则,负载平衡过程结束.

动态负载平衡算法如下.

初始化:定义系统规模,计算每个节点的理想负载和状态,定义负载均衡的起始条件和结束条件.

当(重载节点数 $z>0$)

{如果(重载节点 P_i 中包含的 u_{ij} 的数量 u 大于空载节点数 k 和轻载节点数 q 之和)

在传输时延 η 下,节点 P_i 将每个大小为 u_{ij} 的负载传输给空载节点和轻载节点.

否则,如果(重载节点 P_i 中包含的 u_{ij} 的数量 u 小于等于空载节点数 k)

在传输时延 η 下,节点 P_i 随机选择 u 个空载节点,并将每个大小为 u_{ij} 的负载传输给这 u 个空载节点.

否则

在传输时延 η 下,节点 P_i 随机选择 $u-k$ 个轻载节点,并将每个大小为 u_{ij} 的负载传输给这 $u-k$ 个轻载节点和所有空载节点.

}

更新节点的信息,输出结果.

3 实验结果与分析

在 Intel Core i7 平台上对 CloudSim 仿真框架做出扩展并进行实验. CloudSim 是墨尔本大学 Rajkumar Buyya 教授团队开发的云计算仿真器,是一个通用、可扩展的新型仿真框架,支持无缝建模和模拟,对不同应用和服务模型的调度和分配策略的性能进行量化的比较,达到控制使用云计算资源的目的[11].

实验环境:Win8 操作系统,JDK1.8 版本,CloudSim3.0.3 版本,Eclipse 4.7.1a. 根据实验要求,对主机和 VM 相关参数进行配置,如表 1 所示.

表 1 主机和 VM 参数描述

Table 1 Host and VM parameters

参数	描述
hostId	主机 ID,分别为 1,2,3,4.
CPU	Intel Core i7 -4712MQ
hostram	主机内存,8 GB
hostmips	主机每秒执行百万条指令数,在 2.66 GHz 下对应 49000 DMIPS.
Vmid	虚拟机 ID,分别为 1,2,3,4.
vmmips	根据式(1)计算每个虚拟机的处理能力,再转化为其对应的 MIPS.
vmram	虚拟机内存,分别为 4,2,4,2 GB.
bw	带宽为 1000 Mb/s

按照 CloudSim 的实验步骤,在创建

DataCenter 和 PowerDataCenter 时对其进行扩展，实现所需的任务调度算法.

为了便于分析系统中各节点状态及队列长度的变化，以 4 个节点为例进行分析说明. 4 个节点的初始队列分别为 $w_1=420$，$w_2=980$，$w_3=1500$，$w_4=0$. 图 2 描述了各个节点的状态及队列长度的变化.

从图 2 中可以看到，各节点状态及队列长度变化发生在 $T1=17.723$ ms，$T2=22.366$ ms，$T3=25.001$ ms，$T4=27.431$ ms 和 $T5=29.057$ ms 时刻. 在初始时刻 $T0$，统计每个节点的瞬时负载信息和节点状态，并判断系统状态，根据式(2)对理想负载进行处理；各节点线程计算本节点处理能力 a_i 和理想负载 w_i'. 初始时刻各节点分别处于适载、重载、轻载和空载状态. 节点 2 分别给节点 3 和节点 4 迁移一个单位的负载 $u_2(T1)$. 迁移后在 $T1$ 时刻节点 4 转变为轻载状态. 但是由于节点 3 的处理能力远强于节点 2，所以接收到的一个单位的负载 $u_2(T1)$ 后不会对其状态产生影响. 节点 2 继续给节点 3 和节点 4 迁移一个单位的负载 $u_2(T2)$. 由于节点 2 迁出单位变小，因此在 $T2$ 时

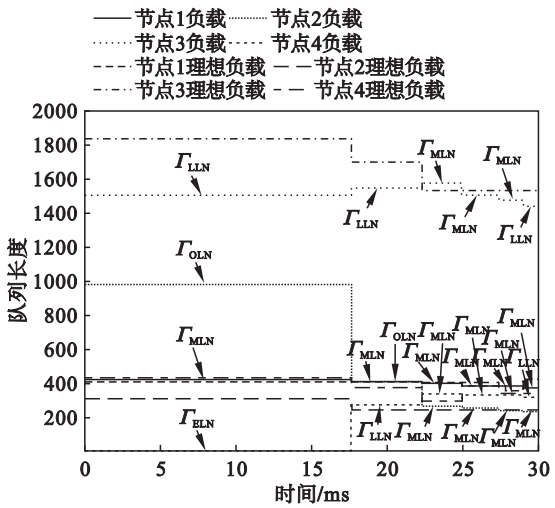

图 2　负载平衡过程中节点的队列长度和状态变化

Fig. 2　State and queue length change of nodes in the process of load balancing

刻的 w_2 落差明显低于 $T1$ 时刻，此时没有轻载和空载节点. 所以重载节点 2 不迁移负载. 所有节点均处理自己的负载. 到 $T3$ 时刻，节点 1 和 4 转变为轻载状态. 节点 2 开始分别给这 2 个节点迁移一个单位的负载 $u_2(T4)$. 在 $T4$ 时刻，4 个节点变为重载、适载、适载和适载状态. 由于 $u_1(T4)$ 升高导致处理能力 a_1 降低，从而使 w' 降低，因此节点 1 成为了重载节点. 各节点均处理本节点的负载不迁移也不接收. 到 $T5$ 时刻，系统中不存在重载节点，达到平衡状态.

为了证明本模型和算法的有效性，不仅与传统算法中的均分负载算法和二分负载算法进行比较，还与使用典型的控制模型实现负载平衡的 DDLB 算法[1]进行比较，系统的负载平衡时间如图 3 所示.

从图 3 中可以看出，在不同规模下，本模型算法所需的负载平衡时间均比其他算法少. 二分负载算法中，因为每次按网络中的节点数将网络划分为两个子网络，然后按照两个子网络的处理速度之比将负载进行迁移，递归上述过程，直到系统达到平衡状态，该过程对负载多次重复迁移，产生较大的数据传输代价. 与其比较，本模型负载平衡时间减少了 62.21%. 均分负载算法和 DDLB 算法均没有考虑负载迁移节点与其他节点的处理能力，也没有考虑负载接收节点的状态，导致过量负载被多次迁移，但是与二分负载算法相比迁移次数减少很多. 与均分负载算法和 DDLB 算法相比，本模型负载平衡时间分别减少了 18.11% 和 9.14%.

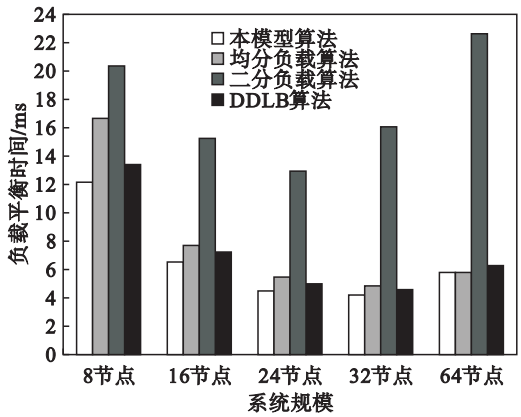

图 3　几种负载平衡算法比较

Fig. 3　Comparison of different load balancing algorithms

负载平衡过程中，在负载进行迁移时所产生的数据传输开销对系统最终达到负载平衡所需的时间影响较大. 图 4 为几种负载平衡算法的数据传输代价的比较图.

从图 4 中可以看出，本模型所采用的算法在各种规模下的数据传输开销都是最小的，这是因为在负载迁移过程中，以每个节点的处理能力和节点当前所处的状态为依据，将重载节点的负载按照合适的比例迁移到相应的节点上，既不会将负载迁移到其他重载节点上，同时尽量避免了负载的重复迁移，降低了系统的数据传输代价. DDLB 算法中每个重载节点以增益 k 将过量负载

按固定划分比例平均迁移给其他所有节点,同时接收其他节点迁移出的过量负载,导致系统中每个节点频繁迁出迁进负载,使重载节点负担加重,但是它重复迁移的负载比均分负载和二分负载算法少.

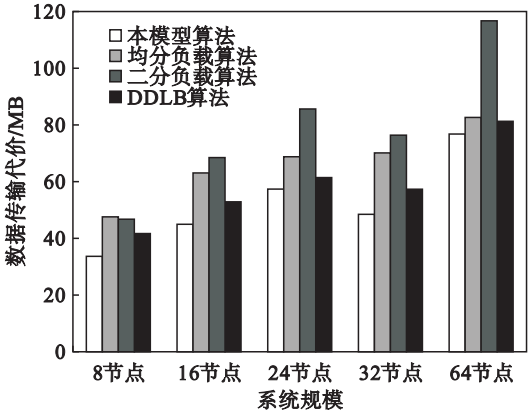

图 4 数据传输代价

Fig. 4 Data transmission cost

4 结 论

本文采用控制理论思想对分布式动态负载均衡过程提出了一个基于状态空间描述的时滞脉冲切换负载平衡模型. 对系统中每个节点的不同状态建立对应的子系统. 在进行负载迁移时,综合考虑本节点和其他所有节点的处理能力,不会导致迁移后出现更多的重载节点,有效平衡节点负载,提高了负载均衡的效率. 通过实验分析,本模型算法使负载平衡时间平均减少 29.82% . 今后将采用 Lyapunov 泛函方法对本系统进行稳定性分析,分析其对负载平衡的影响.

参考文献:

[1] Meng Q Y, Qiao J Z, Lin S K, et al. A delay-based dynamic load balancing method and its stability analysis and simulation[C] // Proceedings of European Conference on Parallel Computing. Ischia, 2010: 192 – 203.

[2] Shah R, Veeravalli B, Misra M. On the design of adaptive and decentralized load-balancing algorithms with load estimation for computational grid environments[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2007, 18(12): 1675 – 1686.

[3] Xu J, Lam A, Li V. Chemical reaction optimization for task scheduling in grid computing[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2011, 22(10): 1624 – 1631.

[4] Jiang Y C, Zhou Y F, Li Y P. Reliable task allocation with load balancing in multiplex networks[J]. *ACM Transactions on Autonomous and Adaptive Systems*, 2015, 10(1): 1 – 32.

[5] Kang Q M, He H, We J. An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems[J]. *Journal of Parallel and Distributed Computing*, 2013, 73(8): 1106 – 1115.

[6] Faragardi H R, Shojaee R, Keshtkar M A, et al. Optimal task allocation for maximizing reliability in distributed real-time systems [C]//Proceedings of the IEEE/ACIS 12th International Conference on Computer and Information Science. Niigata, 2013: 513 – 519.

[7] Subrata R, Zomaya A, Landfeldt B. Game-theoretic approach for load balancing in computational grids [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2008, 19(1): 66 – 76.

[8] Bajaj R, Agrawal D. Improving scheduling of tasks in a heterogeneous environment [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2004, 15(2): 107 – 118.

[9] Jiang Y C, Li Z F. Locality-sensitive task allocation and load balancing in networked multiagent systems: talent versus centrality [J]. *Journal of Parallel and Distributed Computing*, 2011, 71(6): 822 – 836.

[10] Jiang Y C, Zhou Y F, Wang W Y. Task allocation for undependable multiagent systems in social networks[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2013, 24(8): 1671 – 1681.

[11] Kumar R, Sahoo G. Cloud computing simulation using CloudSim[J]. *International Journal of Engineering Trends and Technology*, 2014, 8(2): 82 – 86.