云计算环境下基于蚁群优化算法的资源调度策略

刘 永[1],王新华[1,2],邢长明[3],王 硕[1]
(1. 山东师范大学 信息科学与工程学院,山东 济南 250014;
2. 山东省分布式计算机软件新技术重点实验室,山东 济南 250014;
3. 山东财政学院 继续教育学院,山东 济南 250014)

摘 要:针对当前云计算环境中节点规模巨大,单个节点资源配置较低,寻找有效计算资源效率不高的缺点,文中在Google公司的Map/Reduce框架上提出了两个基于蚁群优化的资源调度策略ACO1和ACO2,并在这两个资源调度策略中引入双向蚂蚁机制。在该双向蚂蚁机制中蚂蚁通过相互交流,能够快速地发现合适的虚拟机资源,从而使得Master节点能够快速地为用户任务分配虚拟机。实验结果表明这两个利用了双向蚂蚁机制的资源调度策略显著减少了为用户任务寻找虚拟机的时间,从而使得用户任务能够更快地获得虚拟机,保证用户作业能够按时完成。

关键词:云;资源调度;蚁群算法
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2011)09-0019-05

Resources Scheduling Strategy Based on Ant Colony Optimization Algorithms in Cloud Computing

LIU Yong[1], WANG Xin-hua[1,2], XING Chang-ming[3], WANG Shuo[1]
(1. Dept. of Information Science and Engineering, Shandong Normal University, Jinan 250014, China;
2. Shandong Provincial Key Laboratory for Distributed Computer Soft Novel Technology, Jinan 250014, China;
3. School of Further Education, Shandong University of Finance, Jinan 250014, China)

Abstract: It presents two resources scheduling algorithms which are named ACO1 and ACO2 respectively for the cloud computing because of the disadvantage that the scale of nodes is huge, the configuration of nodes is not high and the efficiency of finding nodes is low. The two resources scheduling algorithms are based on ant colony algorithm and Map/Reduce frame which belongs to Google's company. And two-way ant mechanism is introduced into the two resources scheduling algorithms. In the mechanism the ants can find the virtual machines which perform the tasks fast by the communication of ants so that the Master node can assign the virtual machines to the tasks fast. The experimental result demonstrates that the time to find virtual machines which perform the tasks by ACO1 and ACO2 reduces observably, which advantage of the two-way ant mechanism so that it reduces the time to assign the virtual machines to the tasks and assures the users' job can be completed on time.

Key words: cloud; resources scheduling; ant colony algorithm

0 引 言

云计算是从并行计算、分布式计算、效用计算等概念发展而来,它通过付费使用的方式向用户提供所需要的各种服务。云计算系统对用户来讲是透明的,其本质是通过虚拟化的计算和存储资源池进行动态部署、动态分配/重新分配、实施监控,从而向用户提供满足QS要求的计算服务、数据存储服务以及平台服务[1]。

云计算具有支持虚拟化、提供服务质量保证、高可靠性、可扩展性、自治性等特征[1]。云计算正深刻地改变着人们对软件的认识和使用模式。人们无需再安装、维护软件,通过付费的方式就可以从互联网上获得想要的服务。就像人们使用水、煤气、电一样方便。云是由大规模的廉价的计算节点构成,像Google公司的云就是由网络连接起来的几十台甚至上百万台的廉价计算机构成[2],由于这些节点受计算、存储、宽带能力等限制,研究云计算资源调度策略对于缩短为用户任务寻找合适的虚拟机资源,加快把用户任务分配到合适的虚拟机上,保证用户作业按时完成,从而满足公司

收稿日期:2011-02-11;**修回日期**:2011-05-16
基金项目:山东省优秀中青年科学家科研奖励基金(BS2010DX032)
作者简介:刘 永(1985-),男,山东济南人,硕士研究生,研究方向为云计算、网格计算;王新华,博士,教授,研究方向为云计算、车载网络;邢长明,博士,研究方向为网格计算;王 硕,硕士研究生,研究方向为云计算、车载网络。

与用户签订的服务等级协议 SLA(Service Level Agreement)具有十分重要的意义。

1 相关工作

目前云计算中的用户作业调度策略和资源调度策略研究的比较多。对于云中的作业调度,Buyya 提出了一个云环境下的 Meta-scheduling Model 模型和 Meta-scheduling policies 策略[3],基于该模型和作业调度策略,可以保证在完成用户作业的同时,使整个数据中心的电量消耗或二氧化碳的排放量最小;Buyya 等人[4]还利用云来扩展集群以应对本地集群资源匮乏,并使用一些作业调度策略(如 Naive, Shortest Queue, Weighted Queue 等)来评估作业性能、分析资源成本。实验结果表明在负载很重的情况下,选择 Naive 调度策略来使用云中的资源将产生很大的成本,而使用基于扩展因子的 request backfilling 和 redirection 策略来使用云中的资源,成本则相对低一些;对于云中的资源调度,吕良干[5]提出了基于信任驱动的资源负载均衡调度算法。该算法考虑了用户任务的信任需求,以资源负载均衡为目标,同时兼顾任务执行时间跨度等因素。模拟结果表明,该算法能提高资源的负载均衡度,减少任务的相对执行时间。

蚁群优化算法是20世纪90年代 M. Dorigo 最早提出的,并用于解决计算机中经典的旅行商问题[6]。随后便出现了大量改进的蚁群优化算法,并应用到 job-shop 问题[7]、网络路由问题[8]、车载路由问题[9]等各种问题中去。目前,大多数蚁群算法中的蚂蚁主要通过信息素交流,蚂蚁之间没有直接的交流,因此算法的收敛性不高。考虑到云计算环境和网格环境有较大差异,云的规模较大,各个节点的计算能力比网格站点资源的计算能力稍低,但是服务质量并不低于网格计算,因此高效的资源调度算法是提高云计算性能的关键。

因此文中提出了两个资源调度策略 ACO1 和 ACO2,并在这两个资源调度策略中引入了双向蚂蚁机制,希望通过蚂蚁的直接交流,可以促进云中可用计算资源的快速发现,从而提高算法的收敛性,保证公司与用户签订的服务等级协议 SLA(Service Level Agreement)。

2 基于蚁群算法的资源调度策略

2.1 双向蚂蚁机制

2.1.1 双向蚂蚁的定义

把蚂蚁分为两类:前向蚂蚁 Forward-Ant 和反向蚂蚁 Back-Ant。前向蚂蚁用于寻找云中可用虚拟机节点(在下文中也称为节点或计算节点);反向蚂蚁在前向蚂蚁找到可用资源后产生,按原路返回,并在途中经过的各节点上留下可用资源的信息素、任务预计执行时间等信息。前向蚂蚁和反向蚂蚁的结构分别如表1和表2所示。

表1 前向蚂蚁结构

蚂蚁标识	路径节点集
AntID	N_s

表2 反向蚂蚁结构

蚂蚁标识	路径节点集	任务预计执行时间	有效节点信息素
AntID	N_s	ET_e	τ_e

2.1.2 蚂蚁相遇处理

在现实生活中当蚂蚁觅食时,若两只蚂蚁相遇,会通过触角进行交流,然后一只蚂蚁会根据另一只蚂蚁提供的线索能很快找到食物[10]。在蚂蚁搜索资源时增加了蚂蚁相遇处理,用以提高算法的收敛速度。

首先,在各节点上预留一块存储区,用于存储反向蚂蚁携带的信息,并启动一个定时器。若定时器归零前,前向蚂蚁到达该节点,就认为两只蚂蚁相遇。归零时,节点会自动清除信息。

一个节点在不同时刻可能会有多个反向蚂蚁经过,它们可能由多个可用资源产生。文中有两种方法解决。

第一种方法是一个节点保存的区域只能存放一个反向蚂蚁携带的信息。当后一个反向蚂蚁到达时,将覆盖前一个反向蚂蚁留下的信息,并重新启动定时器。这种方法只考虑了保留一个可用资源的信息,优点是节点保留的空间小,在计算下一跳节点的概率时计算量比较少,缺陷是没有保留所有可用资源的信息。

第二种方法是一个节点保存的区域能够存放多个反向蚂蚁携带的信息。只要有反向蚂蚁经过,就为该蚂蚁启动一个定时器。并且若两只反向蚂蚁来自同一个可用资源,节点只保存后来的反向蚂蚁携带的信息。这种方法的优点是把所有的可用资源都考虑到了,缺点是节点保留的空间大,计算量大。上面两种方法分别对应 ACO1 和 ACO2 两种资源调度算法。将在2.4.2节中对这两种算法分别进行讨论,并通过实验比较它们的收敛性。

2.2 信息素的定义及修改

2.2.1 信息素的定义

受文献[11]的启发,用虚拟机的硬件资源来衡量一个节点的信息素。主要用 CPU 个数 m,处理能力 p(MPIS),内存容量 r,外存容量 h,带宽 b 来衡量节点的信息素。并按公式(1)为各参数设置阈值,若超过该阈值,则统一以阈值计算。

$$m_{max}=m_0, p_{max}=p_0, r_{max}=r_0, h_{max}=h_0, b_{max}=b_0 \quad (1)$$

先初始化节点的各硬件信息素:

$$\text{CPU 的信息素:}\ \tau_{ij}(0)=\frac{m*p}{m_0*p_0}\times 100\% \tag{2}$$

$$\text{内存的信息素:}\ \tau_{ir}(0)=\frac{r}{r_0}\times 100\% \tag{3}$$

$$\text{外存的信息素:}\ \tau_{ih}(0)=\frac{h}{h_0}\times 100\% \tag{4}$$

$$\text{带宽的信息素:}\ \tau_{ib}(0)=\frac{b}{b_0}\times 100\% \tag{5}$$

节点 i 的信息素是上面各个信息素的带权和,如公式(6)所示:

$$\tau_i(0)=a\tau_{ij}(0)+b\tau_{ir}(0)+c\tau_{ih}(0)+d\tau_{ib}(0),\ a+b+c+d=1 \tag{6}$$

2.2.2　信息素的修改

信息素的修改分为两类,一类是对可用资源节点上信息素的修改,另一类是对反向蚂蚁所经过的各节点上保存的反向蚂蚁所携带的有效节点的信息素的修改。

(1)有效节点信息素的修改。

当新任务被分配到计算节点上时,CPU 利用率等会增大,信息素减小。规定当新任务分配到计算节点上时,信息素按公式(7)更新。

$$\tau_i(t_1)=\tau_i(t)-\lambda\tau_i(t),0<\lambda<1 \tag{7}$$

$\tau_i(t)$ 表示计算节点在时刻 t 的信息素浓度,$\tau_i(t_1)$ 是新任务在 t_1 时刻到达节点 i 的信息素浓度,λ 是调节因子。

当任务在 t_2 时刻运行完或失败时,系统负载减轻。为了保证节点负载平衡,此时信息素浓度按公式(8)增加。

$$\tau_i(t_2)=\tau_i(t_1)+\lambda_1\tau_i(t_1),0<\lambda_1<1 \tag{8}$$

$\tau_i(t)$ 表示时刻 t_2,节点 i 上的信息素浓度。λ_1 是调节因子。

为了奖励成功完成任务的节点,增加该节点的信息素浓度,以使更多的前向蚂蚁选择该节点。任务失败的节点也要予以惩罚。因此在上式的基础上,又添加了一个因子 λ_2,来增加或减少节点的信息素浓度,见公式(9)。

$$\tau_i(t_2)=(1+\lambda_2)(\tau_i(t_1)+\lambda_1\tau_i(t_1)),0<\lambda_1<1 \tag{9}$$

若任务成功运行,$0<\lambda_2<1$;

否则 $-1<\lambda_2<0$。

(2)各个节点保存的信息素的修改。

随着时间的推移,有效节点上的任务数越来越少,负载越来越轻,节点的信息素递增。因此在各节点上保存的有效节点的信息素也要递增。因此每隔一段时间,各节点就要按公式(10)增大保存的有效节点的信息素浓度。

$$\tau_e(t+1)=\tau_e(t)+\lambda_3\tau_e(t),0<\lambda_3<1 \tag{10}$$

$\tau_e(t)$ 表示 t 时刻各节点保存的有效节点 e 的信息素浓度。$\tau_e(t+1)$ 表示 $t+1$ 时刻各节点修改后的有效节点 e 的信息素浓度。λ_3 是调节因子。

2.3　任务预计执行时间的定义及修改

2.3.1　任务预计执行时间的定义

在云计算环境中,一个节点可能要同时运行多个任务。但是在云里把任务分配给效率最高的节点将会提高整个云的性能。因此建立了一个任务预计执行时间模型来预测新任务在节点上的执行时间,如公式(11)所示。

$$ET_j^{n\text{---}}(J_{predict}(t_2))=\frac{n_{predict}}{n_{previous}}\times\begin{pmatrix}\rho ET_j^{n\text{---}}(J_{previous}(t_0))+\\(1-\rho)\,RT_j^{n\text{---}}(J_{previous}(t_1))\end{pmatrix} \tag{11}$$

$ET_j^{n\text{---}}(J_{predict}(t_2))$ 表示新任务 $J_{predict}$ 在 t_2 时刻估计在 j 节点上的预计执行时间,$n_{predict}$ 表示 t_2 时刻 j 上的负载,这里用 j 运行的任务数表示负载大小。$ET_j^{n\text{---}}(J_{previous}(t_0))$ 表示 t_0 时刻上一个完成的任务 $J_{previous}$ 达到 j,在 j 上的预计执行时间。$RT_j^{n\text{---}}(J_{previous}(t_1))$ 表示上一个在 t_1 时刻完成的任务 $J_{previous}$ 的实际执行时间。$n_{previous}$ 表示上一个任务运行时的负载。

2.3.2　任务预计执行时间的修改

因为有效节点随着时间运行的任务越来越少,有效节点上任务预计完成的时间也就越来越少。因此每隔一段时间用公式(12)修改各节点上保存的有效节点上的任务预计执行时间。

$$ET_e^{n\text{---}}(J_{predict}(t+1))=(1-\rho_1)\times ET_e^{n\text{---}}(J_{predict}(t)),0<\rho_1<1 \tag{12}$$

2.4　前向蚂蚁选择下一跳节点规则

2.4.1　前向蚂蚁与反向蚂蚁没相遇

如果前向蚂蚁没有遇到反向蚂蚁,就按公式(13)算,选择其中概率大的节点为下一跳节点。

$$p_{ij}=\frac{\tau_j^{\alpha}/A_j^{\beta}}{\sum_{m\notin N_i}(\tau_m^{\alpha}/A_m^{\beta})},j\notin N_i \tag{13}$$

$A_j=ET_j^{n\text{---}}(J_{predict}(t))$,$p_{ij}$ 是前向蚂蚁在节点 i 选择下一跳节点 j 的概率。τ_j 是节点 i 观察到节点 j 上的信息素浓度。N_i 是前向蚂蚁的路径节点集。m 是 i 的邻居节点。α,β 为调节因子,分别表示 τ_j 和 A_j 的重要性。

2.4.2　前向蚂蚁与反向蚂蚁相遇

(1)若一个节点只保存一个反向蚂蚁携带的信

息，且反向蚂蚁在节点 i 的前一跳节点为 k，选择下一跳节点时，就要分别计算 P_{ij} 和 P_{ik}。

P_{ij} 中，j 是 i 要选择的下一跳节点且 j 和 k 不是同一个节点。按公式(14)计算。

$$P_{ij}=\frac{\tau_j^{\alpha}/A_j^{\beta}}{\sum_{m\notin N_s}(\tau_m^{\alpha}/A_m^{\beta})+\tau_e^{\alpha}/A_e^{\beta}},\ j\notin N_s\wedge m\neq k \tag{14}$$

P_{ik} 按公式(15)计算。

$$P_{ik}=\frac{\tau_e^{\alpha}/A_e^{\beta}}{\sum_{m\notin N_s}(\tau_m^{\alpha}/A_m^{\beta})+\tau_e^{\alpha}/A_e^{\beta}},\ m\neq k \tag{15}$$

τ_e 为节点 k 保存的有效节点 e 的信息素值，A_e 为 k 节点保存的有效节点 e 的预计执行时间值。

比较 P_{ij} 和 P_{ik} 的大小，前向蚂蚁将选择概率最大的节点为下一跳节点。

(2)若一个节点保存多个反向蚂蚁携带的信息，则要考虑多个有效节点的信息。定义两个集合 $B(K)$ 和 $E(K)$。$B(K)$ 是前向蚂蚁选择下一跳节点恰好是反向蚂蚁经过的节点的集合。$E(K)$ 是各个有效节点的集合。

把下一跳节点 j 分为两类。一类，$j\notin N_s$ 且 $j\notin B(K)$；另一类，$j\notin N_s$ 且 $j\in B(K)$。对于第一类节点，按公式(16)计算。

$$P_{ij}=\frac{\tau_j^{\alpha}/A_j^{\beta}}{\sum_{m\notin B(K)}(\tau_m^{\alpha}/A_m^{\beta})+\sum_{m\in B(K)}(\tau_e^{\alpha}/A_e^{\beta})},\quad m\notin N_s\wedge e\in E(k) \tag{16}$$

对于第二类节点，按公式(17)计算。

$$P_{ij}=\frac{\tau_e^{\alpha}/A_e^{\beta}}{\sum_{m\notin B(K)}(\tau_m^{\alpha}/A_m^{\beta})+\sum_{m\in B(K)}(\tau_e^{\alpha}/A_e^{\beta})},\quad m\notin N_s\wedge e\in E(k) \tag{17}$$

其中两式中概率最大者为下一跳节点。

3 资源调度策略算法描述

资源调度算法 ACO1 和 ACO2 除了第(5)步不同外，其他步骤相同，算法步骤如下：

(1)将各个节点的信息素初始化。

(2)用户向 Master 节点[12]提交一批作业。

(3)Master 节点取第一个作业 j。假定作业 j 的大小是 Z_j，组成作业 j 的任务 t 的任务大小是 k_t。则 Master 节点把作业 j 划分成 Z_j/k_t 个任务。并启动一个定时器，发送 nZ_j/k_t 个前向蚂蚁，其中 n 是一参数，决定前向蚂蚁数量与任务数量的倍数关系。每只前向蚂蚁随机地选择下一跳节点 i。

(4)当前向蚂蚁 Forward-Ant 进入一个虚拟机节点 i 时，将 i 设置到 Forward-Ant 的 N_s 里。并根据公式(11)判断自己是否是有效节点。如果是，则标记为有效节点，并产生一个反向蚂蚁 Back-Ant。将 Forward-Ant 的 N_s 赋给 Back-Ant，使其沿原路返回。假设有效节点为 e，Back-Ant 将携带 e 的信息素，新任务在 e 上的预计执行时间等信息返回。并将这些信息留在经过的各节点上。

(5)若前向蚂蚁 Forward-Ant 在节点 i 上不符合公式(11)且没有遇到反向蚂蚁，则按公式(13)选择下一跳节点。如果遇到了反向蚂蚁，则根据 2.4.2 节中的方法选择下一跳节点。

(6)反向蚂蚁经过的各个节点每隔一段时间按公式(10)修改保留的有效节点的信息素浓度，按公式(12)修改有效节点的任务预计执行时间值。

(7)在 Master 节点启动的定时器归零前，若 Master 收到了反向蚂蚁，则认为产生反向蚂蚁的节点有效，把任务分配到上面，并按公式(7)修改信息素浓度；若定时器超时，Master 还没有收到反向蚂蚁，则认为没有合适的资源，暂不分配任务。

(8)分配到任务的节点，任务完成或失败，节点按公式(9)修改信息素浓度。失败的任务由 Master 节点转移到其他节点来完成。

(9)Master 取下一个作业，重复步骤(3)～(8)。

4 算法仿真结果及分析

为了验证新算法，采用 CloudSim[13]做模拟。并将资源调度策略 ACO1，ACO2 与基于蚂蚁系统 AS 的资源调度策略以及基于蚁群系统 ACS 的资源调度策略在资源调度时间方面进行比较。

4.1 时间复杂度

整个算法的时间复杂度等于 $O(k(m+n))$。其中 k 是作业个数，$O(m)$ 是前向蚂蚁寻找资源的时间复杂度，$O(n)$ 是把作业放到虚拟机资源上的时间复杂度。本算法主要是使前向蚂蚁寻找虚拟机资源的时间尽量减小，从而使得分配虚拟机的整个时间减小。

4.2 实验参数的设置

算法中的 a,b,c,d 分别表示了虚拟机的 CPU，内存，外存及带宽的重要性，因为任务的执行受处理器处理能力的影响较大，因此把 a,b,c,d 的值分别设为 4，2，2，2。α,β 分别表示了信息素，任务预计执行时间的重要性。n 决定了产生多少蚂蚁更加合适。通过实验来获得 α,β,n 的最优组合。调节因子 λ、λ_1、λ_3 都取 0.2；λ_2 在任务成功或失败时分别取 0.2 和 -0.2。ρ 取 0.4，ρ_1 取 0.2。虚拟机节点设为 200 个，Master 的定时器设为 5s。任务大小为 500M，虚拟机处理能力为 200MIPS～400MIPS，内存容量为 512M～1G，带宽为

1M/s ~2M/s,外存容量为10G ~20G。

4.3 实验结果及分析

首先提交一个作业大小为2500M的作业,该作业重复提交10次,最后得出有7次当$\alpha=1,\beta=2,n=2.5$时,为作业任务调度有效节点的时间最少。然后取不同作业大小的作业重复上述过程,最后得出有60%的作业当$\alpha=1,\beta=2,n=2.5$时,为作业任务调度有效节点的时间最少。因此认为$\alpha=1,\beta=2,n=2.5$为最优组合。表3给出了作业大小是2500M,α,β,n取不同值时,资源调度的时间变化。表4给出了作业大小等于2500M,$\alpha=1,\beta=2$, $n=2.5$时,ACO1,ACO2与蚂蚁系统AS以及蚁群系统ACS在资源调度时间方面的比较。

表3　作业大小是2500M, α, β, n取不同值时,资源调度的时间变化

实验	α	β	n	时间
1	1	1	1.5	3.734
2	1	2	1.5	3.725
3	2	1	1.5	3.713
4	1	1	2	3.714
5	1	2	2	3.702
6	2	1	2	3.673
7	1	1	2.5	3.687
8	1	2	2.5	3.612
9	2	1	2.5	3.659
10	1	1	3	3.714
11	1	2	3	3.668
12	2	1	3	3.693

表4　作业大小是2500M, $\alpha=1,\beta=2$, $n=2.5$时,资源调度时间比较

算法	α	β	n	时间
ACO1	1	2	2.5	3.612
ACO2	1	2	2.5	3.604
AS	1	2	2.5	3.713
ACS	1	2	2.5	3.683

从表4可以看出,ACO1,ACO2在资源调度时间方面明显比AS以及ACS用时少,这主要是因为前向蚂蚁搜寻虚拟机的时间减少了,从而使得分配虚拟机的整个时间变短。在前向蚂蚁搜寻虚拟机的过程中增加了蚂蚁相遇时相互交流这一过程,当前向蚂蚁遇到反向蚂蚁时,前向蚂蚁可以根据反向蚂蚁留下的有效节点信息,在有效节点以及其他节点之间做出选择,从而迅速找到适合执行作业任务的节点。ACO2比ACO1花费的时间少,是因为ACO2节点中保存了多个有效节点的信息,前向蚂蚁从这些有效节点中选择一个的概率较大,因此ACO2调度时间比ACO1少。

图1给出了当$\alpha=1,\beta=2,n=2.5$,提交5 ~25个作业,以5为作业跨度时,ACO1,ACO2与蚂蚁系统AS以及蚁群系统ACS在资源调度时间方面的比较。

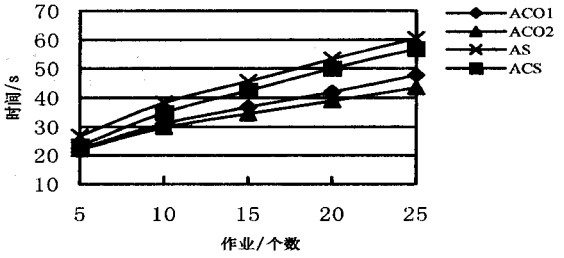

图1　资源调度时间

从图1可以看出,ACO1,ACO2比AS和ACS用时少,这仍然是增加了蚂蚁相遇时相互交流的结果。前向蚂蚁可以根据反向蚂蚁留下的有效节点信息能够快速地寻找到有效的虚拟机节点,从而使得把任务分配到虚拟机上的整个时间变短。

5 结束语

文中提出了两个基于双向蚂蚁机制的资源调度策略ACO1和ACO2。提出的资源调度策略能够适应云计算环境的大规模性、共享性等特征,为用户任务迅速找到合适的虚拟机资源并分配任务,从而保证用户作业按时完成,保证公司与用户签订的服务等级协议SLA。下一步的工作是结合其他算法,进一步加快为用户任务寻找虚拟机资源的收敛性,从而使得把用户任务分配到虚拟机上的时间进一步缩短。

参考文献:

[1] 张建勋,古志民,邓　超.云计算研究进展综述[J].计算机应用研究,2010,27(2):429-433.

[2] 王庆波,金　萍,何　乐,等.虚拟化与云计算[M].北京:电子工业出版社,2009:112-113.

[3] Garg S K, Yeo C S, Buyya R, et al. Energy-Efficient Scheduling of HPC Applications in Cloud Computing Environment [EB/OL]. (2009-09-07). http: //www. cloudbus. org/reports/EE-SchedulingAcrossClouds-2009. pdf.

[4] Assuncao M, Costanzo A, Buyya R. Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters[C]//Proc of the 18th ACM International Symposium on High Performance Distributed Computing. Munich, Germany:[s. n.],2009:141-150.

[5] 吕良干.云计算环境下资源负载均衡调度算法研究[D].乌鲁木齐:新疆大学,2010.

[6] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem[J]. IEEE Transactions on Evolutionary Computation,1997,1(1): 53-66.

[7] Colorni A, Dorigo M, Maniezzo V, et al. Ant system for job-shop scheduling[J]. Belgian Journal of Operations Research, Statistics and Computer Science ,1994,34(1):35-39.

[8] 李领治,郑洪源,丁秋林.一种基于改进蚁群算法的选播路

(下转第27页)

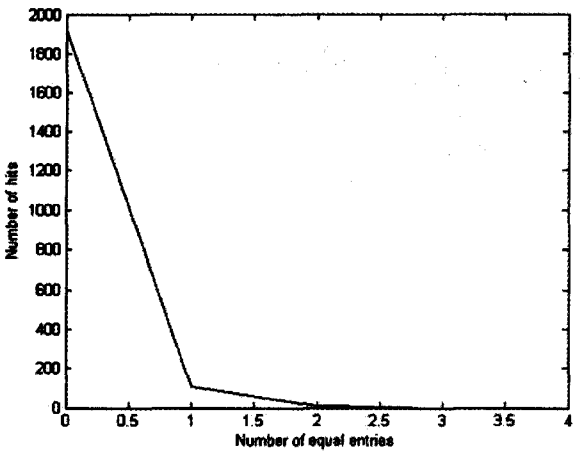

图 5　Hash 值在相同位置上有相同值的 ASCII 码字符的分布图

值的重要标志之一。文中的算法是利用明文消息来构建权重网络,将消息中微小的变化直接扩散到整个网络中,与基于混沌耦合映像格子的 Hash 算法相比,扩散速度更快,从而可以减少混沌运算迭代的次数。然而,本算法是基于混沌系统的 Hash,主要采用浮点数运算,所需算术运算较多,对算法执行效率有所影响。

3　结束语

提出了消息网络的思想,在此基础上构造了单向 Hash 函数。该算法是利用消息明文通过特定的规则构造成具有某种结构的权重网络,并建立消息权重网络与混沌复杂动态网络的映射关系。将消息权重网络的邻接矩阵用到混沌系统中,经过迭代运算,将本次的输出对称交换后作为下个消息块运算的输入,类似的处理所有的消息块,最后一个消息块运算得到的数,经过线性变换和进制转换,得到一定长度的 Hash 值,其中 Hash 值的长度与网络的大小有关。理论分析和数值仿真表明,所提出的新算法具有良好的单向性、置乱性和强的抗碰撞性,增强了消息之间的耦合度,使消息中任何微小的变化都能直接扩散到整个网络中,提高了算法对初始明文的敏感性,降低了算法的复杂度。在网络信息安全领域,该算法可以很容易地完成数字签名、身份认证等功能。

参考文献:

[1] Stallings W. 密码编码学与网络安全——原理与实践[M]. 北京:电子工业出版社,2006.

[2] 郭　雷,许晓鸣. 复杂网络[M]. 上海:上海科技教育出版社,2006.

[3] 刘建东,付秀丽. 基于耦合帐篷映射的时空混沌单向 Hash 函数构造[J]. 通信学报,2007,28(6):30-38.

[4] 张　瀚,王秀峰,李朝晖,等. 基于时空混沌系统的单向 Hash 函数构造[J]. 物理学报,2005,54(9):4006-4011.

[5] 刘光杰,单　梁,孙金生,等. 基于时空混沌系统构造 Hash 函数[J]. 控制与决策,2006,21(11):1244-1248.

[6] 赵　耿,袁　阳,王　冰. 基于交叉耦合映像格子的单向 Hash 函数构造[J]. 东南大学学报(自然科学版),2009,39(4):728-732.

[7] 程艳云,宋玉蓉. 基于耦合映像格子混沌系统的 Hash 函数构造[J]. 应用科学学报,2010,28(1):44-48.

[8] Song Y R, Jiang G P. Constructing hash function based on coupled network generated by logarithmic map[J]. Journal of Nanjing University of Posts and Telecommunications (Natural Science), 2010, 30(1):6-10.

[9] 宋玉蓉,蒋国平. 基于一维元胞自动机的复杂网络恶意软件传播研究[J]. 物理学报,2009,58(9):5911-5918.

[10] Liu J, Chi K T, He K Q. Fierce stock market fluctuation disrupts scale free distribution[J]. Quantitative Finance, 2009, 10(80):7688-7696.

[11] Chen C, Lu J A, Wu X Q. Complex networks constructed from irrational number sequences[J]. Physica A, 2010, 389(13):2654-2662.

[12] Zhang J, Sun J F, Luo X D, et al. Characterizing pseudoperiodic time series through the complex network approach[J]. Physica D, 2008, 237(22):2856-2865.

[13] García P, Parravano A, Cosenza M G, et al. Coupled map networks as communication schemes[J]. Physical Review E, 2002, 65(4):5201-5204.

(上接第 23 页)

由算法[J]. 电子与信息学报,2007,29(2):340-344.

[9] Bell J E, Mcmullen P R. Ant colony optimization techniques for the vehicle routing problem[J]. Advanced Engineering Informatics, 2004, 18(1):41-48.

[10] 潘达儒,袁艳波. 一种基于 AntNet 改进的路由算法[J]. 小型微型计算机系统,2006,27(7):1169-1174.

[11] 张　千,梁　鸿,李　振. 基于改进蚂蚁算法的网格资源管理的研究[J]. 微电子学与计算机,2009,26(9): 71-74.

[12] Yang H C, Dasdan A, Hsiao R L. Map-reduce-merge: Simplified relational data processing on large clusters[C]// Proc of the 2007 ACM SIGMOD International conference on management of data. Beijing, China:[s. n.], 2007:1029-1040.

[13] Calheriros R N, Ranjan R, Buyya R. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services[R]. Melbourne: Grid Computing and Distributed Systems Laboratory, The University of Melbourne, 2009.