

云环境下虚拟机集群系统动态负载均衡机制

李立耀^{1,2*}, 赵少卡¹, 林东森¹, 徐 聪², 杨家海²

(1. 福建师范大学福清分校 数学与计算机科学系, 福建 福清 350300;

2. 清华大学 网络科学与网络空间研究院, 北京 100084)

(* 通信作者电子邮箱 leolee@cernet.edu.cn)

摘 要:针对传统的物理集群系统无法灵活应对大型互联网应用的问题,提出一种云环境下虚拟机集群的综合负载均衡机制。该方法首先定期地采集集群中虚拟机节点的 CPU、内存、连接数、响应时间,以及所在物理主机的负载状况等指标信息,然后加权计算节点的综合负载并得出其权值,最后通过调度器进行任务请求的合理分配,从而解决了传统集群系统负载不均且不能适应多变的网络环境等诸多问题。实验结果表明,与加权轮询法(WRR)和加权最少连接法(WLC)调度方案相比,该机制能够在并发量较大时维持较低的响应时间,并能够根据集群中综合负载的状态实时地增加或减少虚拟机数量,通常在5 s之内达到整体集群的负载均衡。

关键词:云计算;虚拟机;集群;负载均衡;调度

中图分类号: TP393 **文献标志码:** A

Dynamic load balancing mechanism in cloud-based virtual cluster system

LI Liyao^{1,2*}, ZHAO Shaoka¹, LIN Dongsen¹, XU Cong², YANG Jiahai²

(1. Department of Mathematics and Computer Science, Fuzhou Branch of Fujian Normal University, Fuzhou Fujian 350300, China;

2. Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China)

Abstract: As the conventional physical cluster system fails to cope flexibly with large-scale Internet applications, a comprehensive load balancing mechanism for cloud-based virtual cluster system was proposed. It first periodically collected CPU and memory usage, number of connections, and response time of all virtual machines and physical hosts, then calculated the weighted load of the physical hosts, and finally scheduled and assigned the task requests based on the calculated comprehensive load, thus could adapt to the complex, dynamic and variable computing environment. The experimental results show that, compared with other scheduling mechanisms such as Weighted Round Robin (WRR) and Weighted Least Connections (WLC), the proposed mechanism is delay optimal under heavy workload, and moreover, it can increase or decrease the number of Virtual Machines (VMs) dynamically to balance the server load usually within 5 seconds.

Key words: cloud computing; virtual machine; cluster; load balance; scheduling

0 引言

云计算是一种新型网络服务方式,它将硬件资源与软件资源封装为服务,用户可以通过网络按需地访问和使用^[1-2]。随着云技术的快速发展,越来越多的网络应用通过云平台租用虚拟机(Virtual Machine, VM),实现应用部署来提供对外服务,这在一定程度上提高了服务效率并降低了运营成本。一些大型网络应用随着业务的快速发展,将出现大量的并发请求,此时单台的 VM 就不能很好地满足服务的需要,通常将租用多台 VM 组成集群来提高服务能力^[3-5]。

传统集群中,各个任务处理单元实际上就是每个节点的物理主机,为了保证服务质量(Quality of Service, QoS),节点的数量基本是根据预估的用户请求数量的峰值进行确定。由于多数网络应用的保持峰值概率都不会很高,集群的整体负载大部分时间处于低负载的状态,系统资源无法得到充分利

用,造成资源的浪费。另一方面,一些网络应用的发展十分复杂,可能因为某种原因(如营销策略影响)带来大量的用户请求,远远超过原先估算的峰值,引起整个集群负载过高,此时若增加集群节点需要较大的时间成本,无法应对用户请求数量的瞬间变化,造成服务质量降低以及用户流失。

然而虚拟集群系统是由多台 VM 组成一个整体提供服务,初期可以采用 Best-fit、Min-min、随机调度等策略^[6-9]将 VM 较为合理地分配到云平台的各个物理机(Physical Machine, PM)节点上,使得各节点的 VM 在负载上基本均衡。随着时间的推移,PM 中不同 VM 的负载也在不断变化,虚拟集群系统中节点负载不均衡的现象也将随之出现^[10]。为了解决这一问题,可以通过在线迁移(live migrating)将 VM 从重负 PM 节点迁移到其他轻负载 PM 节点上,在此过程中虚拟机还能保持正常运行;但该方法的缺点是在迁移过程本身会占用 PM 的系统资源,且在这个过程中会造成对外服务的片

收稿日期: 2014-06-05; **修回日期:** 2014-08-08。 **基金项目:** 清华大学一思科联合实验室研究基金资助项目(TCJ2013A004);教育部—中国移动研究基金资助项目(MCM20123041);福建省教育厅科技项目(JA12352, JA13343, JB13198)。

作者简介: 李立耀(1970-),男,福建平潭人,副教授,主要研究方向:云计算、计算机网络; 赵少卡(1980-),男,福建福州人,讲师,硕士,主要研究方向:云计算、软件工程; 徐聪(1986-),男,黑龙江东宁人,博士研究生,主要研究方向:计算机网络; 杨家海(1966-),男,浙江云和人,教授,博士,主要研究方向:计算机网络。

刻中断,使系统缺乏稳定性^[11]。而另外一种做法是通过调度用户任务合理分配各 VM,使得 VM 的负载相对均衡,针对这种做法,目前有许多负载均衡调度策略^[12],其中,有些策略属于静态调度方法,如轮询调度、加权轮询(Weighted Round Robin, WRR)调度、目标地址散列调度、源地址散列调度等,它们不能根据具体场景的变化实时动态地调整集群中各节点的权重,可能造成节点间的负载失衡;也有些属于动态调度方法,如最小连接调度、加权最小连接(Weighted Least Connections, WLC)调度等,但是这两种方法只考虑连接数因素,在实际应用中具有局限性。

针对上述问题,本文提出一种基于集群的综合负载计算模型,通过监测集群 VM 节点的负载情况,并使用加权计算进行负载的综合评价,之后根据评价情况实时地对用户请求进行合理有效的分配,并弹性地对集群中 VM 的数量进行增加或减少,最终实现整体集群的动态负载均衡。

1 系统体系结构设计

云环境下虚拟机集群系统(VM Cluster)的体系结构如图 1 所示。 VM_1, VM_2, \dots, VM_n 是分配在 PM 中的虚拟机,其中 $VM_i, VM_j (i < j)$ 可以分布在属于同一台 PM 上,也可部署到不同的 PM 中。

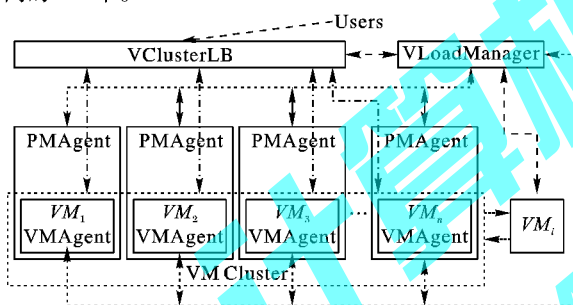


图 1 虚拟机集群系统体系结构

该系统主要由任务调度器(VClusterLB)、负载管理器(VLoadManager)、虚拟机监控代理(VMAgent)、物理机监控代理(PMAgent)等子系统所组成。

PMAgent 与 VMAgent 负责监控集群中相应设备的 CPU 利用率、内存利用率、网络 I/O、磁盘 I/O 等相关参数,并将收集到的信息发送给 VLoadManager 进行处理。

VLoadManager 负责根据监测信息计算集群各节点的权重,并把计算结果发送给 VClusterLB,而且在平台运行的任意时刻,都可以根据集群的负载情况增加或减少 VM 的数量,从而实现整个集群实时动态的负载均衡。它既可以直接部署在物理节点上,也可以部署在虚拟机或虚拟机集群中。

VClusterLB 负责云计算中心虚拟集群的调度,它接收用户的访问请求,根据 VLoadManager 传来的权值计算结果及本身调度策略,并根据各个节点处理能力的不同对提交的用户请求进行分配,将请求调度到集群的各个节点,决定由哪台或哪些 VM 最终为用户提供服务,使得集群内部各个节点处理的请求数目处于相对均衡状态。

2 VM Cluster System 实现

用户向云平台提交所需要集群系统的要求,如硬件需求(CPU、内存、块设备等)、软件需求(操作系统、服务软件等)

以及任务要求(最大请求并发量、最长响应时间),以结构化文档保存到后台数据库表^[13]。

之后,系统根据用户的需求在各物理节点生成虚拟机,并把它们组成集群。具体可以简单描述成两个过程:

1) 虚拟机镜像模板文件的生成。

首先使用虚拟机管理软件(如基于内核的虚拟机(Kernel-based Virtual Machine, KVM)或 Xen 等)创建镜像文件并用它启动虚拟机,接着安装操作系统以及集群需要的一些专用软件,如代理监控软件(VMAgent)、提供为客户端服务的软件、负载均衡器软件(VClusterLB)等,最后把安装好的系统作为镜像模板上传到云中心的镜像管理服务器。

2) 虚拟机部署。

虚拟机部署包括虚拟机生成过程与虚拟机集群过程:虚拟机的生成过程是根据用户提交的需求,由云中心的资源调度服务模块根据策略选择物理节点作为集群虚拟机节点的宿主机,然后从镜像服务器拷贝集群所需的镜像到该物理节点,从而生成虚拟机;接着进行虚拟机的集群,通过进一步设置各虚拟机所属的虚拟网络、安全组等,将虚拟机的各种参数加以设定,保证虚拟机节点之间正常通信。同时根据分配的 IP 地址、MAC 地址等信息由资源调度中心自动地对虚拟机集群的中 VClusterLB 应用服务进行相应配置,从而得到一个完整的虚拟机集群。

其中,VClusterLB 是整个集群系统的调度中心,一般独立于其他节点进行部署。外部访问的流量都会经过它再转交到其他节点处理,当流量超过它所能承受的上限,会成为整个集群的瓶颈,集群的 QoS 就会受到影响。本文采用弹性机制对 VClusterLB 进行部署,当它的负载超过上限时,请求资源调度中心再分配一台虚拟机进行部署,解决调度器的瓶颈。

3 综合负载方案设计

3.1 参数指标的收集与预处理

为了更好地记录节点各种负载信息以及处理服务的性能情况,本文方案收集了各节点 CPU 使用率($CPU_Utilization_i$)、内存利用率($MEM_Utilization_i$)、当前节点连接数(N_i)、响应时间($RESPONSE_TIME_i$)等指标。其中, $RESPONSE_TIME_i$ 的值由 VLoadManager 节点向各节点轮询请求相应的服务得到, $CONNECT_i$ 指标由 VClusterLB 收集,而其他指标的信息则由 VMAgent 与 PMAgent 周期性(时间周期为 T)收集并定时发送给 VLoadManager 节点。

为了统一地表示各参数指标,对 $CONNECT_i$ 与 $RESPONSE_TIME_i$ 进行预处理。

具体地,设 $N_i (i = 1, 2, \dots, n)$ 是节点 VM_i 在 t 时刻的连接数,则 $\sum_{k=1}^n N_k$ 是所有节点的总连接数。将 $CONNECT_i = N_i / \sum_{k=1}^n N_k$ 表示为节点 VM_i 的连接参数指标。

同理,节点 N_i 的响应时间为 $RESPONSE_TIME_i$,则 $\sum_{k=1}^n RESPONSE_TIME_k$ 是所有节点的响应时间总和。将节点 VM_i 的响应时间参数指标 $TIME_SCALE_i$ 表示为:

$$TIME_SCALE_i = \frac{RESPONSE_TIME_i}{\sum_{k=1}^n RESPONSE_TIME_k}$$

3.2 节点综合负载的定义

为了更好地反映各种参数对集群节点负载的影响,定义节点的综合负载 $Load_{VM_i}$,具体做法如下:

$$Load_{VM_i} = R_1 \times CPU_Utilization_i + R_2 \times MEM_Utilization_i + R_3 \times CONNECT_i + R_4 \times TIME_SCALE_i + R_5 \times Load_{PM_i} \quad (1)$$

其中: $Load_{PM_i}$ 是指 VM_i 节点所在的物理节点服务器的负载; $R_i (i = 1, 2, \dots, 5)$ 是各种参数指标的权值,且 $\sum R_i = 1$ 。显然 $Load_{VM_i} \in [0, 1]$, $R_i (i = 1, 2, \dots, 5)$ 的值可以根据具体运行的服务程序测得。

3.3 节点的权值计算

在集群系统初始运行时,对于每个节点 VM_i 可以设定一个初始的权值 $WEIGHT_i$,传入负载均衡器 VClusterLB 进行初始态的调度。之后,随着节点 VM_i 负载的变化,不断对权值进行适当调整。

令权值调整范围为 $[WEIGHT_{min}, WEIGHT_{max}]$, 设 $Current_Weight_i$ 为 VM_i 节点的当前权值,该权值可以根据各节点虚拟处理器 (Virtual CPU, VCPU) 的数量、内存大小、网络带宽以及所属的物理节点性能进行定义 (本文只根据 VCPU 的数量、内存大小定义), VLoadManager 根据式 (1) 计算出综合负载 $Load_{VM_i}$ 。设集群节点的最佳综合负载为 $Load_{optimal}$,一般值为 $0.7 \sim 0.8$ 。系统调整后新负载的权值用 $WEIGHT_{new}$ 表示,引入一个策略算法,定义如下:

$$WEIGHT_{new} = \begin{cases} \max\{WEIGHT_{min}, (1 - \delta_1) \times Current_Weight_i\}, & Load_{VM_i} > Load_{optimal} + \Delta \\ Current_Weight_i, & Load_{VM_i} \in [Load_{optimal} - \Delta, Load_{optimal} + \Delta] \\ \min\{WEIGHT_{max}, (1 + \delta_2) \times Current_Weight_i\}, & Load_{VM_i} < Load_{optimal} - \Delta \end{cases} \quad (2)$$

其中: Δ 是常量,表示 $Load_{VM_i}$ 在一定区域范围内允许误差,主要是保证负载的相对稳定;而 δ_1, δ_2 用于调节收敛速度,通常取 $\delta_1 > \delta_2$ 。当节点负载高于 $Load_{optimal}$ 时,能够较快地降低负载;当节点负载低于 $Load_{optimal}$ 时,较平滑地逐渐增加负载量,以提高集群整体的服务质量。从式 (2) 可看出 $\delta_i (i = 1, 2)$ 值越大收敛速度越快,所以 δ_i 需要取合理的值, $Load_{VM_i} > Load_{optimal} + \Delta$, 当前节点的权值 $WEIGHT_{new}$ 会逐渐减小,而 $Load_{VM_i} < Load_{optimal} - \Delta$ 时,当前节点的权值 $WEIGHT_{new}$ 会逐渐增加,因此式 (2) 最终将使得权值调整到一个较为稳定的点。

特别地,若某节点响应时间 $RESPONSE_TIME_i$ 超过设定的时间阈值 $RESPONSE_TIME_{max}$ 时,表明该节点没有响应,无法接受任务,此时将权值设置为 0,用户请求的任务就不会分配给该节点,直到下一次有响应时,再次对该节点的权值进行计算调整。

在实际系统中,若发现所有节点的权值 $Current_Weight_i$ 都接近于 $WEIGHT_{min}$, 即 $|Current_Weight_i - WEIGHT_{min}| \leq \lambda$ (λ 取很小值,本文取 0.5), 或 $Current_Weight_i = 0$ 则说明整个虚拟集群超负载,这时应考虑可以增加节点来降低整体负载;反之,若发现所有节点的权值都接近于 $WEIGHT_{max}$, 则说

明整个虚拟集群负载较轻,这时可以考虑减少节点数,最终实现整体负载相对合理,具体算法流程见图 2 所示。

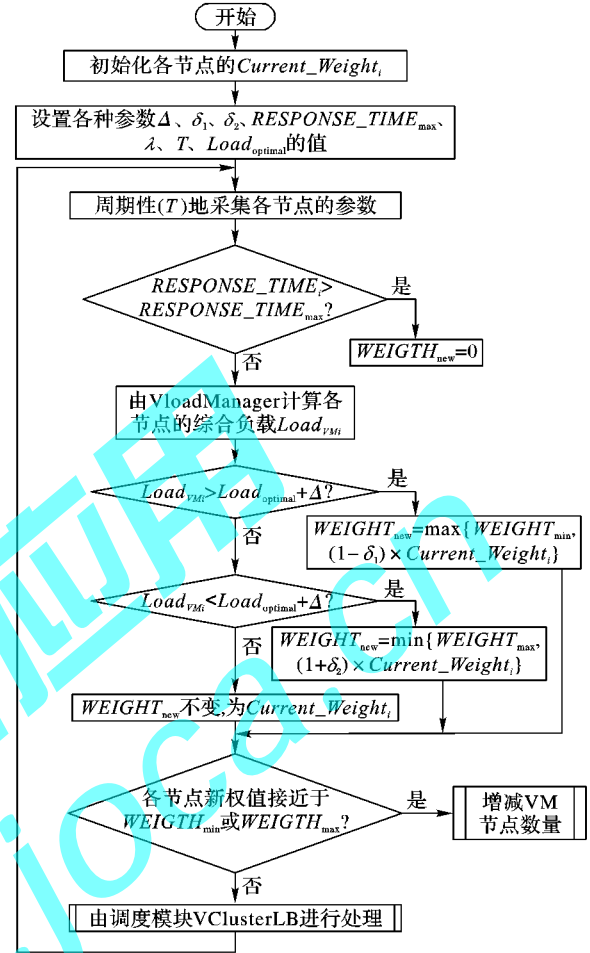


图2 动态负载均衡机制算法流程

4 实验与分析

为了验证该动态均衡机制的可行性,本文基于 OpenStack (havana 版) 构建了一个云计算 IaaS 平台^[15], 并进行模拟实验。

实验平台环境由 12 台服务器、1 个存储系统、10 台客户机及千兆网络组成,硬件配置如表 1 所示,其中每台服务器装有 Ubuntu12.04 并选用 KVM 虚拟化技术进行虚拟机管理;客户机安装有 Windows 7 桌面版系统及 ab、webbench 等压力测试软件。

表1 测试服务器配置

项目名称	配置信息	数量
服务器 1	Xeon CPU E5-2689 @ 2.60 GHz, 128 GB 内存, 2 块硬盘 300 GB, 4 张千兆网卡	3
服务器 2	Xeon CPU E5-2620 @ 2.00 GHz, 64 GB 内存, 2 块硬盘 300 GB, 4 张千兆网卡	5
服务器 3	CPU i5-3470 3.2 GHz, 内存 8 GB, 2 张千兆网卡	4
存储系统	48 TB, 4 张千兆网卡	1

利用上述平台,搭建一个虚拟机集群,集群由 32 个节点、1 台安装有 VClusterLB 模块的 VM 和 1 台安装有 VLoadManager 模块的 VM, 以及 30 台计算节点所组成,各 VM 的资源配置见表 2。该集群提供了一个基于 Apache 服务的

Web 应用,该应用服务包含了静态网页文件(A类)、计算任务(B类)、内存负载(C类)等三类文件,通过 webbench 工具对该集群进行多组压力测试,经过统计对比,确定式(1)中的权值 $\{R_1, R_2, R_3, R_4, R_5\}$ 为 $\{0.2, 0.05, 0.24, 0.43, 0.08\}$ 。

表 2 测试虚拟集群配置

项目名称	配置信息	数量
lvs-VClusterLB	4 GB 内存,4 虚拟内核	1
lvs-VloadManager	4 GB 内存,4 虚拟内核	1
lvs-tiny	512 MB 内存,1 虚拟内核	5
lvs-small	1 GB 内存,1 虚拟内核	10
lvs-medium	2 GB 内存,2 虚拟内核	10
lvs-large	4 GB 内存,4 虚拟内核	5

在该虚拟机集群中,VloadManager 模块、VMAgent 模块以及 PMAgent 模块的实现主要是调用 Openstack 的 ceilometer 组件的 API 以及 libvirt 所提供的 virt-top 工具,并结合 shell 脚本、C 语言,最终使用 python 语言加以实现,各模块代码已经整合到部署环境中。

实验一 集群压力测试。

为了测试集群的负载状况,实验选择静态调度(加权轮询调度)策略与动态调度(加权最小连接调度)策略与本文调度策略进行比较,对相关参数进行取值,其中, $\delta_1 = 0.2, \delta_2 = 0.15, \Delta = 0.05, Load_{optimal} = 0.7$ 。此外,将表 2 中 lvs-tiny、lvs-small、lvs-medium、lvs-large 的虚拟节点权值分别设定为 1, 2, 4, 8, 则权值区间 $[WEIGHT_{min}, WEIGHT_{max}]$ 可设为 $[1, 8]$ 。

实验中的负载测试由 webbench 工具提供,按照一定规则对 A 类、B 类与 C 类文件进行请求访问,对预先设置的并发量实施每隔 20 s 增加 500 个的策略,测得集群的平均响应时间如图 3 所示。

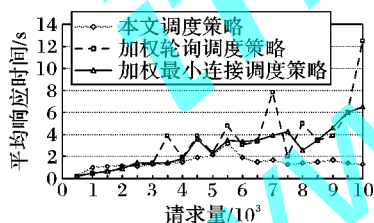


图 3 集群平均响应时间

从该实验中不难看出,在负载较低的情况下,集群中的计算节点需要采集各种指标参数,因此本文策略与其他调度方案相比相差不大。但随着并发量的增大,集群用本文调度方案的响应时间抖动幅度不大,保持相对稳定,而加权轮询调度和加权最小连接调度方案抖动范围较大,主要原因是当负载增加时,集群动态地对请求进行更合理的分配,并在过载时适时地增加节点,从而使整体集群的负载调节到合理的水平,而其他方法在运行过程中出现节点负载倾斜,后续的请求被调度给负载较重节点时,其响应时间就会受到影响,实验体现了本调度策略的优势。

实验二 集群弹性测试。

为了测试集群的弹性,依然利用实验一的相关参数,在初始时,集群的并发量较小,处于低负载状态,维持一段时间后,采集各项参数指标并计算得到集群的平均负载值 $Load_{avg}$ 。之后,当观察到 $Load_{avg}$ 的值增大时,开始增加并发量,让系统处于高负载状态,维持一段时间后,采用相同的方法计算

$Load_{avg}$,测得集群的 $Load_{avg}$ 变化状况如图 4 所示。

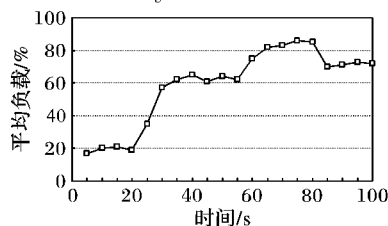


图 4 集群平均负载

从本实验可看出:保持低负载状态一定时间后,由于减少了集群中的 VM,使得整体的平均负载 $Load_{avg}$ 有所增加。同样地,保持高负载状态一定时间后,由于增加了集群中的 VM,使得整体的平均负载 $Load_{avg}$ 得以减少。从图 4 可看出:整体平均负载可以保持在 70% 左右,集群中各节点的资源利用率得到充分利用;集群能够在较短时间(大约 5 s)内完成 VM 的增加或减少,实现集群负载的弹性变化。

5 结语

本文提出了云环境的一种虚拟机集群系统动态负载均衡机制,通过一定的时间间隔,监测集群中虚拟机节点的各项指标信息及所在物理节点的相关指标,并加权计算节点综合负载值,使得集群能够根据具体场景的变化动态地调整各节点的权重,从而对外部请求任务进行合理的分配,还可以根据综合负载状态实时地对集群虚拟机数量进行弹性的增加或减少,一定程度上实现整体集群的动态负载均衡,并保证各节点资源的充分利用。但是,如何将本策略应用于更大范围的异构环境并加以验证还有待于进一步的研究。

参考文献:

- [1] RAJKUMAR B, SHIN Y C, VENUGOPAL S, *et al.* Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. *Future Generation Computer Systems*, 2009, 25(6): 599–616.
- [2] LUO J, JIN J, SONG A, *et al.* Cloud computing: architecture and key technologies[J]. *Journal on Communications*, 2011, 32(7): 3–21. (罗军舟, 金嘉晖, 宋爱波. 云计算: 体系架构与关键技术[J]. *通信学报*, 2011, 32(7): 3–21.)
- [3] DU Y, GUO T, CHEN J. Fleet elastic load balancing mechanism in cloud environment[J]. *Journal of Computer Applications*, 2013, 33(3): 830–833. (杜焱, 郭涛, 陈俊杰. 云环境下机群弹性负载均衡机制[J]. *计算机应用*, 2013, 33(3): 830–833.)
- [4] JIAO Y, WANG W. Design and implementation of load balancing of distributed-system-based Web server[C]// *Proceedings of the 2010 Third International Symposium on Electronic Commerce and Security*. Piscataway: IEEE Press, 2010: 337–342.
- [5] GILLY K, JUIZ C, PUIGJANER R. An up-to-date survey in Web load balancing[J]. *World Wide Web*, 2011, 14(2): 105–131.
- [6] SPEITKAMP B, BICHLER M. A mathematical programming approach for server consolidation problems in virtualized data centers[J]. *IEEE Transactions on Services Computing*, 2010, 3(4): 266–278.
- [7] QIN X, XIE T. An availability-aware task scheduling strategy for heterogeneous systems[J]. *IEEE Transactions on Computers*, 2008, 57(2): 188–199.

(下转第 3090 页)

况下,MS-PSO 的运行时间会随着通信时间的增加而线性增加。图 8(b)给出了 MS-PSO 在不同适应值计算时间下,加速比随着问题维数的改变而变化情况。由式(5)可知,在进化操作时间与适应值计算时间固定的情况下,随着通信时间的增加,加速比会有所减小,图 8(b)中的 4 条加速比曲线的变化趋势符合这一分析。但值得注意的一点是,当适应值计算时间占的比重不同时,通信开销变化带来的影响不同。从图 8(b)可看出:当适应值计算时间较短时,加速比曲线下降的斜率较大,即是说,通信开销增加所带来的性能影响较大;相对地,在适应值计算时间较长的情况下,MS-PSO 对通信开销的容忍度也较大。因此,在分布式进化算法求解大规模问题时,如果适应值计算时间所占的比重较大,则可认为算法对通信开销变化具有一定的容忍度,不会因为通信开销的增加引起加速比的急剧下降。

5 结语

本文介绍了分布式进化算法的实现工具、常用模型,引入并分析了影响算法性能的三个因素。为了验证进化操作开销、适应值计算开销以及通信开销这三个因素的实际影响,本文展开了三部分实验。实验结果表明,当适应值计算开销占总开销的比重较大时,分布式进化算法能够展现良好的性能,此时,通信开销的上下浮动不会引起算法加速比的急剧变化。本文所包含的性能因素分析以及展开的各项实验可以作为分布式进化算法的设计者的设计参考。

参考文献:

- [1] TANG T, WEI L, XIE X, *et al.* Study on distributed parallel computing on hybrid genetic algorithm[J]. *Computer Engineering and Applications*, 2011, 47(9): 207 - 209. (唐天兵, 韦凌云, 谢祥宏, 等. 分布式并行计算环境下混合遗传算法的研究[J]. *计算机工程与应用*, 2011, 47(9): 207 - 209.)
- [2] MIAO Q, KONG Z, WANG Y. Function optimization based on distributed immune evolutionary algorithm[J]. *Systems Engineering and Electronics*, 2012, 34(2): 413 - 417. (苗启广, 孔哲鹏, 王艳红. 基于分布式免疫进化算法的函数优化[J]. *系统工程与电子技术*, 2012, 34(2): 413 - 417.)
- [3] LASSIG J, SUDHOLT D. The benefit of migration in parallel evolutionary algorithms[C]// GECCO 2010: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. New York: ACM Press, 2010: 1105 - 1112.
- [4] LI C, WANG J, YAN X, *et al.* Subpopulation diversity based accepting immigrant in distributed evolutionary algorithms[C]// IC-PADS 2013: Proceedings of the 2013 International Conference on Parallel and Distributed Systems. Piscataway: IEEE Press, 2013: 422 - 423.
- [5] LASSIG J, SUDHOLT D. Adaptive population models for offspring populations and parallel evolutionary algorithms[EB/OL]. [2013-10-10]. <http://www.icsi.berkeley.edu/pubs/algorithms/adaptive-populationmodels11.pdf>.
- [6] WHITLEY D. A genetic algorithm tutorial[J]. *Statistics and Computing*, 1994, 4(2): 65 - 85.
- [7] BRATTON D, KENNEDY J. Defining a standard for particle swarm optimization[C]// SIS 2007: Proceedings of the 2007 Swarm Intelligence Symposium. Piscataway: IEEE Press, 2007: 120 - 127.
- [8] ALBA E, TOMASSINI M. Parallelism and evolutionary algorithms[J]. *IEEE Transactions on Evolutionary Computation*, 2002, 6(5): 443 - 462.
- [9] VITELA J E, CASTANOS O. A real-coded niching memetic algorithm for continuous multimodal function optimization[C]// CEC 2008: Proceedings of the 2008 IEEE Congress on Evolutionary Computation. Piscataway: IEEE Press, 2008: 2170 - 2177.
- [10] ZHAN Z, ZHANG J, LI Y, *et al.* Adaptive particle swarm optimization[J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2009, 39(6): 1362 - 1381.
- [11] YAO X, LIU Y, LIN G. Evolutionary programming made faster[J]. *IEEE Transactions on Evolutionary Computation*, 1999, 3(2): 82 - 102.
- [12] ZHANG J, CHUNG H, LO W, *et al.* Implementation of a decoupled optimization technique for design of switching regulators using genetic algorithm[J]. *IEEE Transactions on Power Electronics*, 2001, 16(6): 752 - 763.
- [13] DUBREUIL M, GAGNE C, PARIZEAU M. Analysis of a master-slave architecture for distributed evolutionary computations[J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2006, 36(1): 229 - 235.
- [8] ZHAO S, LI L, LING X, *et al.* Architecture and scheduling scheme design of Tsinghua cloud based on OpenStack[J]. *Journal of Computer Applications*, 2013, 33(12): 3335 - 3338. (赵少卡, 李立耀, 凌晓, 等. 基于 OpenStack 的清华云平台构建与调度方案设计[J]. *计算机应用*, 2013, 33(12): 3335 - 3338.)
- [9] MAGULURI S T, SRIKANT R, YING L. Stochastic models of load balancing and scheduling in cloud computing clusters[C]// Proceedings of the 2012 IEEE INFOCOMM. Piscataway: IEEE Press, 2012: 702 - 710.
- [10] MAGULURI S T, SRIKANT R, YING L. Heavy traffic optimal resource allocation algorithms for cloud computing clusters[EB/OL]. [2013-10-10]. <http://arxiv.org/pdf/1206.1264v1.pdf>.
- [11] CHEN Y, HUAI J, HU C. Live migration of virtual machines based on hybrid memory copy approach[J]. *Chinese Journal of Computers*, 2011, 34(12): 2278 - 2291. (陈阳, 怀进鹏, 胡春明. 基于内存混合复制方式的虚拟机在线迁移机制[J]. *计算机学报*, 2011, 34(12): 2278 - 2291.)
- [12] ZHANG W. LVS cluster load dispatch[EB/OL]. [2013-10-10]. <http://www.linuxvirtualserver.org/zh/lvs4.html>. (章文嵩. LVS 集群的负载调度[EB/OL]. [2013-10-10]. <http://www.linux-virtualserver.org/zh/lvs4.html>.)
- [13] FENG L, FU Y, CHEN K, *et al.* TDDS: task deployment and scheduling based on virtual cluster system[J]. *Journal of Computer Research and Development*, 2013, 50(5): 1118 - 1124. (冯琳, 付勇, 陈康, 等. TDDS: 基于虚拟集群系统的任务部署与调度[J]. *计算机研究与发展*, 2013, 50(5): 1118 - 1124.)
- [14] OpenStack installation guide for Ubuntu 12.04[EB/OL]. [2013-12-20]. <http://docs.openstack.org/havana/install-guide/install/appt/content/>.

(上接第 3085 页)