第 41 卷 第 2 期 计 算 机 工 程 2015 年 2 月
Vol. 41 No. 2 Computer Engineering February 2015

·云计算专题· 文章编号:1000-3428(2015)02-0007-05 文献标识码:A 中图分类号:TP302.1

基于事件驱动机制的虚拟化故障检测恢复系统

崔竞松[1a,1b],路昊宇[1a],郭 迟[2],何 松[1a]

(1. 武汉大学 a. 计算机学院;b. 空天信息安全与可信计算教育部重点实验室,武汉 430072;
2. 武汉大学卫星导航定位技术研究中心,武汉 430079)

摘 要: 为解决虚拟化条件下云平台故障排除不及时的问题,在开源云平台 OpenStack 上设计并实现一种虚拟化故障检测恢复系统。该系统由 GUI 层、调度层、逻辑层和功能层组成,以事件驱动机制为核心,将系统中传递的信息作为事件按时序进行处理。以感知模块、策略模块、执行模块为主体,调用 OpenStack API 和 Libvirt API 实现与虚拟机管理层的交互。建立以信息获取、分析处理、故障恢复为主要内容的故障检测恢复体系,通过对云平台运行环境的实时检测,获取状态参数,根据策略对参数进行分析判断并制定应对措施,实现对故障的自动恢复。实验结果证明,该系统可以在无代理情况下对云平台进行实时检测和故障自动恢复,增强云环境的安全性,提升云平台的高可用性。

关键词: OpenStack 云平台;负载均衡;事件驱动机制;高可用性;虚拟化;云计算

中文引用格式: 崔竞松,路昊宇,郭 迟,等. 基于事件驱动机制的虚拟化故障检测恢复系统[J]. 计算机工程,2015,41(2):7-11,16.

英文引用格式: Cui Jingsong,Lu Haoyu,Guo Chi,et al. Virtualization Fault Detection Recovery System Based on Event-driven Mechanism[J]. Computer Engineering,2015,41(2):7-11,16.

Virtualization Fault Detection Recovery System Based on Event-driven Mechanism

CUI Jingsong[1a,1b],LU Haoyu[1a],GUO Chi[2],HE Song[1a]

(1a. School of Computer Science; b. Key Laboratory of Aerospace Information and Trusted Computing, Wuhan University,Wuhan 430072,China; 2. Global Navigation Satellite System Research Center, Wuhan University,Wuhan 430079,China)

【Abstract】 In order to solve the problem that the fault troubleshooting of cloud platforms is not timely,and guarantee the continuity of cloud services,this paper designs and implements a virtualization fault detection and recovery system based on event-driven mechanism,which is on the open-source cloud platform——OpenStack. The system is composed of GUI layer,scheduling layer,logic layer and functional layer,and processes the information transmitted in the system by timing as an event on the basis of event-driven mechanism. It mainly uses perception module,policy module and execution module,which call OpenStack API and Libvirt API to interact with the management of virtual machines. The established fault detection recovery system mainly includes information acquisition,analysis and processing,fault recovery,and by real-time detection of the cloud platform's runtime environment,it can obtain state parameters,analyze the parameters and develop countermeasures according to established policy,and achieve automatic fault recovery. Experimental results show that the system can detect and recover cloud platforms' fault with agentless method,enhance the security of cloud environments,and improve the high availability of cloud platforms.

【Key words】 OpenStack cloud platform; load balancing; event-driven mechanism; high availability; virtualization; cloud computing

DOI: 10.3969/j.issn.1000-3428.2015.02.002

基金项目: 国家"863"计划基金资助项目(2013AA12A206);国家自然科学基金资助项目(41104010,91120002,61170026);中央高校基本科研业务费专项基金资助项目(2042014kf0237)。

作者简介: 崔竞松(1975-),男,副教授、博士,主研方向:信息安全,云安全;路昊宇,硕士研究生;郭 迟(通讯作者),讲师、博士;何 松,硕士研究生。

收稿日期: 2014-03-21 **修回日期:** 2014-05-09 **E-mail:** guochi@whu.edu.cn

1 概述

云计算将大量的计算资源、存储资源与软件资源链接在一起,形成巨大规模的共享虚拟 IT 资源池[1],综合分布式计算、虚拟化技术、负载均衡技术、网络存储技术等传统计算机技术[2],屏蔽了底层的差异,把不同的、独立的计算机资源抽象成统一的、共享的资源统一向外界提供服务[3]。在云计算环境下,IT 领域按需服务的理念得到了真正体现,云计算通过整合分布式资源,构建应对多种服务要求的计算环境,满足用户定制化要求,并可通过网络访问相应的服务资源[4]。

云平台资源的高度集中化使得其高可用性变得非常重要,任何系统维护与宕机都可能引起较大规模的服务缺失[5]。对于一个计算中心而言,提供一种高可用性的能力服务于所有在其上运行的应用是要考虑的关键问题之一[6]。现有的云平台都提供了一定的技术保证了云的高可用性,比如 VMware vSphere[7] 能够检测虚拟机所在物理主机运行情况,但是对运行中的虚拟机及应用程序方面的监测较少,一旦发生故障将不能得到及时处理,导致服务中断。本文在 OpenStack 基础上设计并实现一种基于事件驱动机制的虚拟化故障检测恢复系统,从物理机、虚拟机、虚拟机应用 3 个层次对云平台环境进行监控,实时获取运行参数,实现对故障的无代理检测恢复。

2 系统设计

OpenStack[8] 是美国国家航空航天局和 Rackspace 公司共同开发的开源云计算平台,旨在为公共及私有云的建设与管理提供各种组件。系统在 OpenStack 基础上,通过 API 调用整合形成 GUI 层、Scheduler[9] 调度层、逻辑层和功能层,并按逻辑将功能层和逻辑层划分为感知模块、策略模块和执行模块三大核心模块,采用事件驱动机制将各部分有机结合起来,通过对事件的处理实现系统对故障的检测和恢复功能。系统架构如图 1 所示。

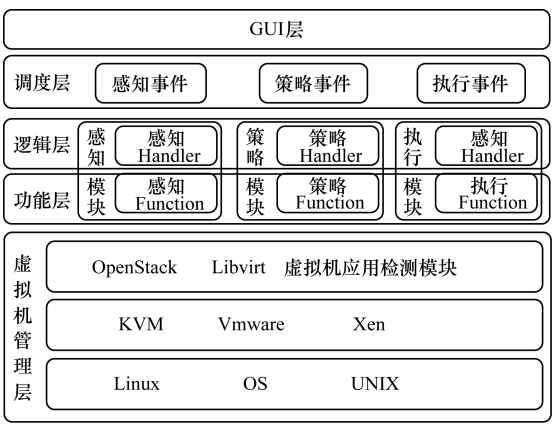

图 1 系统架构

(1) 功能层:通过对 Libvirt API[10] 和 OpenStack API 的封装调用,实现 OpenStack 和 Libvirt 进行交互,连接系统和虚拟机管理层,同时功能层作为整个系统的最底层为其它各层的函数调用及相关功能提供支持。

(2) 逻辑层:采用事件驱动机制,将系统中传递的信息规范为事件由相应的对象进行处理。事件主要分为感知事件、策略事件和执行事件。

(3) 调度层:负责对事件队列进行操作,作为系统运行的枢纽进行调度,保证事件有条不紊地被执行,对事件队列满、事件队列空等各种异常进行处理。

(4) GUI 层:用来与用户进行交互,用户可以通过该界面,获得系统的运行参数,主要包括虚拟机的运行状态、虚拟机的资源占用等情况。同时,用户可以通过界面控制虚拟机的运行状况(比如停止某台虚拟机),对系统的部分基本配置(比如打捞时间的设置等)进行更改。

(5) 核心模块:三大模块均由对应的 Handler 和 Function 组成。感知模块负责感知整个系统的运行情况;策略模块根据感知的结果进行处理;执行模块负责执行策略模块制定的恢复策略,共同构成系统的检测恢复核心。

3 事件驱动机制

为了实现不同层面、不同模块、不同进程(线程)之间通信和信息处理,系统采用事件驱动的方式,将感知到的信息、制定的策略等系统中传递的信息作为事件,由事件处理进程进行处理,平时事件处理进程处于等待状态,由接收到的任何一个事件进行驱动[11]。事件驱动主要设计 3 个类:Event 事件类,Handler 逻辑类,Scheduler 调度类。在事件驱动机制中,Event 是派生所有事件的基类,用来标识系统中各种行为,比如启动虚拟机;Handler 是派生所有事件处理对象的基类,用来处理相应 Event 事件类;Scheduler 调度类负责调度事件,事件队列 Queue 定义在 Scheduler 类中,Scheduler 对事件队列进行维护,用来保证事件被有序处理。事件驱动机制总体设计如图 2 所示。

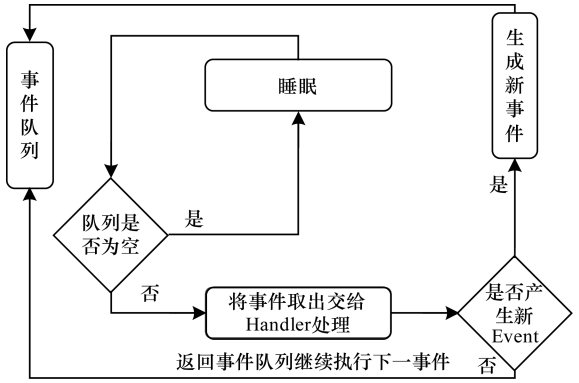

图 2 事件驱动机制总体设计

事件驱动的过程主要分 3 个步骤:首先是将事件放入全局的事件队列 Queue 中,然后按时间顺序从事件队列中取出事件,再通过指定的事件处理对象(即 Handler)处理事件,在处理过程中可能会往事件队列中加入新的事件。这样不断取出事件并处理一直到事件队列为空,就实现了一次感知过程。

Scheduler 主要负责事件的调度,采用基于事件[12]的周期轮询机制,按时序对系统中的事件进行统一的调度和处理。事件队列是由优先级和时间决定的有序队列,按照优先级由高到低的顺序执行,在优先级相同的情况下按照事件执行时间先后排序执行,确保队首永远是第一个要执行的事件。Scheduler 采取每执行一个事件就在后台启动一个线程的方式防止事件等待加快并行,将队列设置成静态变量防止多线程队列,实现对队列的高度维护。由于事件的插入和取出并执行相互独立,可能会发生队列中第一个事件还没到执行时间,Scheduler 正在休眠,实然插入一个需要立即执行的新事件需要唤醒队列,为此引入锁机制。Scheduler 休眠需满足2 个条件:(1)执行时间不小于现在系统的时间;(2)锁必须是关闭的。当 Scheduler 休眠时,新到达的事件执行时间小于现在系统时间,系统就会打开锁,从而破坏 Scheduler 休眠的必要条件,唤醒 Scheduler。

4　关键技术

系统的主体是由功能层和逻辑层共同组成的感知模块、策略模块和执行模块,在此基础上通过事件驱动机制进行调度,实现对故障的自动检测和恢复功能,具体工作流程如图 3 所示。

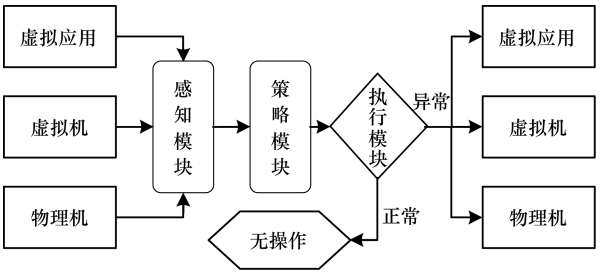

图 3　系统工作流程

通过感知模块获取系统运行信息,由策略模块对感知的信息进行分析并区别制定策略,执行模块根据策略进行故障恢复、提示系统管理员或者在系统运行正常的情况下无动作。从而在云平台中某台物理服务器发生故障时,能够自动检测故障并选择合适的物理机重建运行在该物理机上所有虚拟机;当物理机上某台虚拟机发生故障时,能够删除故障虚拟机,根据其快照或镜像快速进行重建;当虚拟机中的某个应用进程发生故障时(进程非正常退出、非正常挂起),在无代理的情况下,直接操作 GuestOS 内核对象,恢复启动虚拟机里的进程。

4.1　感知模块

感知模块由感知 Function 和 Handler 组成,其中,Function 主要封装 Libvirt API 获取信息实现底层功能,完成对上层 Handler 的支持。感知模块 Function 类结构如图 4 所示。

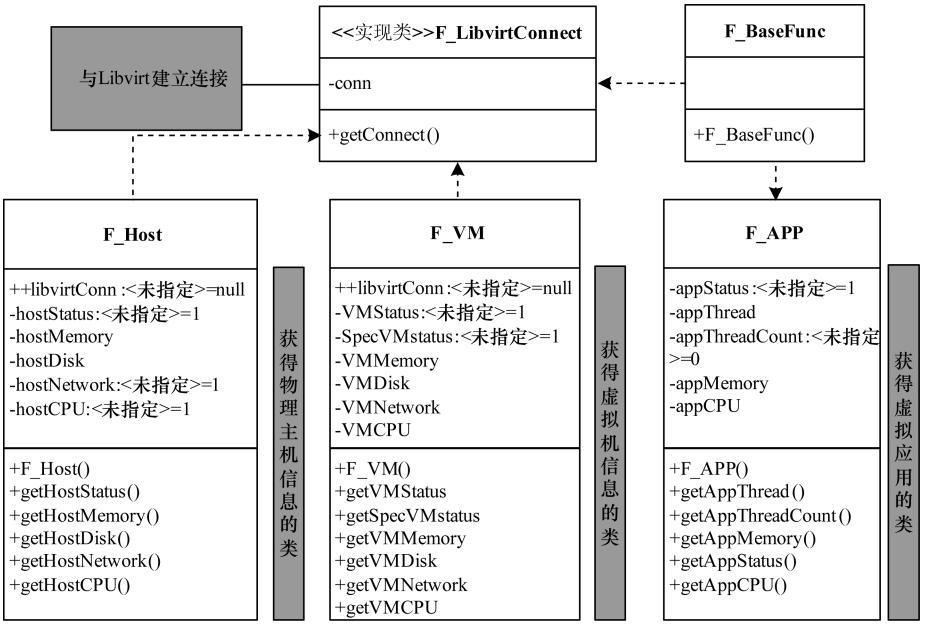

图 4　感知模块 Function 类

感知模块通过感知 Function 类中的 F_Host、F_VM、F_APP 3 个子类分别获取物理机、虚拟机、虚拟机应用的信息,由相应的 Handler 类进行处理,生成对应的感知事件进入策略模块进行处理。其中,物理机主要考察主机状态、内存、CPU、网络、磁盘等信息,虚拟机主要考察全局(指定)虚拟机状态、网络、磁盘、内存等信息,虚拟机应用主要考察进程状态、进程数、内存等信息。以物理机状态获取为例,

Function 类获取信息流程为:(1)确定获得物理主机状态的类 F_hostStatus;(2) 通过与主机建立 Libvirt 连接调用函数 getConnect( ) 获得与物理机的连接 conn;(3)对 conn 进行判断,若 conn 为 fals 则相应的物理机状态为 0(0 表示物理机断电),若 conn 为 ture,则相应物理机状态为 1(1 表示物理机正常)。

4.2 策略模块

策略模块对感知模块传送来的感知事件进行分析,根据策略机制来判断平台出现何种故障,并生成相应的策略事件。策略机制是整个模块的核心,不同的事件对应不同的策略机制。以恢复虚拟机的机制为例,共有 3 张表需要维护,第 1 张表 Instance_Info,信息主要来自 nova 的数据库并由 nova 进行维护,保存虚拟机的基本信息。它是一个字典的结构,每一项也是一个字典,格式为{虚拟机的序列号:{'id':' ','isActive':' '}}。第 2 张表 VM_State,主要由 Libvirt 维护,它的信息主要是感知模块感知到的信息,它也是一个字典结构,每一项为{虚拟机的 id:虚拟机的状态}。第 3 张表 Execution_State,记录一个虚拟机正在进行恢复的操作,其信息主要是根据Instance_Info表和VM_State表,信息的修改主要由执行 Handler 来执行。恢复虚拟机的逻辑机制如表 1 所示。

表 1 虚拟机恢复的逻辑机制

Instance_Info	VM_State	Execution_State	措施	备注
Active	Active	0	无操作	虚拟机无操作
Active	Not Active	1	修复虚拟机	虚拟机正在执行,不可再操作
Active	无	1	修复虚拟机	VM_State 无此表项
Not Active	无	1	修复虚拟机	
无	Active	无	通知管理员有非法用户	Execution_State 要和 Instance_Info 保持一致,无该表项
无	Not Active	无	通知管理员有非法用户	

4.3 执行模块

执行模块主要是对策略模块分析制定的策略响应并采取相应措施,对故障进行恢复。执行模块 Function 类图如图 5 所示。

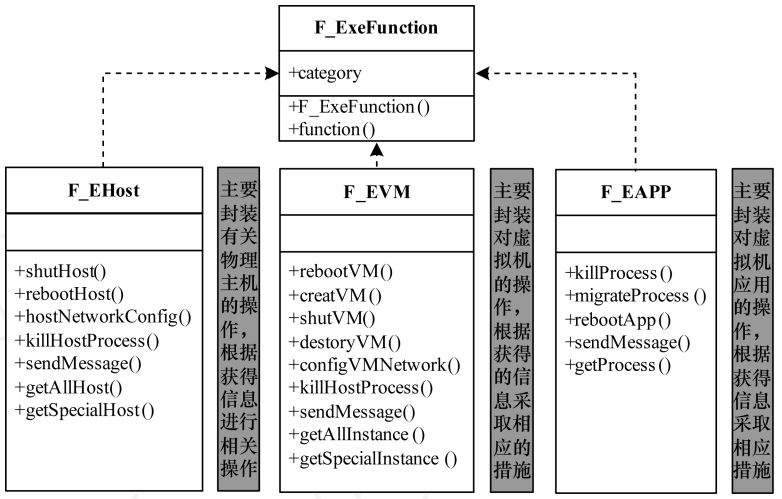

图 5 执行模块 Function 类图

F_ExeHost 主要处理物理主机异常,可以对物理机进行关闭、重启、配置网络、杀死进程、向管理员发送异常报告的操作;F_ExeVM 主要处理虚拟机出现异常,可以重新开启一台虚拟机,对运行的虚拟机进行关闭、销毁、配置网络、向管理员发送异常报告的操作;F_EAPP 主要处理虚拟应用出现异常,包括杀死进程、重启应用程序、向管理员发送异常报告等。执行 Handler 类的核心是 OpenStack API 的调用。Handler 类通过调用 OpenStack 平台提供的对外接口,用类与 OpenStack 进行交互,利用 OpenStack 的现有功能完成操作。

4.4 负载平衡

为了在创建和恢复虚拟机时实现负载均衡、提高资源利用率,由 Nova-Scheduler[13] 分析并选择性能最佳的物理主机创建并启动虚拟机,共分 3 个步骤:

(1) 主机过滤:在获取到集群中所有物理主机列表的情况下,根据特定的属性过滤掉不满足条件的主机,生成新的主机列表。过滤器可以使用 OpenStack 提供的默认过滤器,也可以根据需要进行设置,创建特定的过滤器。

(2) 权值计算:将过滤后的主机列表进行性能评估。在此采取权值算法,给主机的每个特性设定权重值,将内存大小、磁盘容量、网络流量等特性值分别相应的权重相乘,把得到的数值加起来就得到主机的综合权值。

(3) 主机选择:根据每台主机的综合权值对列表

内的主机进行排序,从而选择合适的主机进行虚拟机的创建与重启。根据虚拟机的不同特性要求,也可以选择主机的部分特性进行权值计算后进行排序。

5　实验结果与分析

5.1　实验环境

实验平台由 5 台主机组成,其中主控节点 1 台,主要配置 nova, glance, keystone, horizon, mysql 服务,向计算节点发送命令;共享存储 1 台,主要配置 oracle 服务,具有数据库、共享存储功能;计算节点 3 台(分别为 B1, B2, B3),主要配置有 nova-compute,nova-network 服务,运行虚拟机提供虚拟应用。计算节点 3 台主机运行的虚拟机分别为 VM1, VM2, VM3,操作系统依次为 Windows Xp Sp3, Ubuntu 12.04 LTS和 Windows 7 Sp2。实验环境配置如表 2 所示。

表 2　实验环境配置

项目	主控节点	共享存储	计算节点
处理器型号	Inter Ⓡ E8400	Inter Ⓡ E8400	Inter Ⓡ E8400
处理器核数	2	2	2
主频	3.0 GHz	3.0 GHz	3.0 GHz
内存容量	4.00 GB	4.00 GB	4.00 GB
磁盘容量	160 GB	1 TB	160 GB
操作系统	Ubuntu 12.04 Server	Windows 7 Sp2	Ubuntu 12.04 Server
Linux 内核版本	3.2.0-60	–	3.2.0-60
Qemu-kvm 版本	–	–	1.4.0
Libvirt 版本	1.0.0	–	1.0.0
OpenStack 版本	OpenStack Folsom	–	OpenStack Folsom

5.2　实验测试

实验测试过程如下:

(1) 物理主机故障:在物理主机 B1 上正在运行有虚拟机 VM1,因为 B1 突然断电不能正常工作,B2,B3 均正常工作,系统检测到后重建了虚拟机 VM1,并向管理员发送异常报告。在其他条件不定的情况下改变 B2、B3 的硬盘大小,虚拟机在 2 台物理主机上重建的概率分布如图 6 所示。

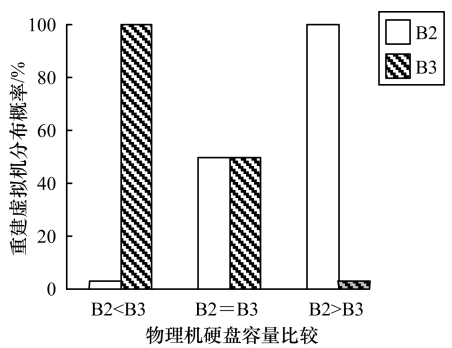

图 6　物理主机选择分布

(2) 虚拟机故障:在计算节点的物理主机 B1, B2,B3 上分别运行有虚拟机 VM1,VM2,VM3,对以下 2 种情形测试:1)在终端利用 Libvirt 命令删除虚拟机导致虚拟机崩溃,系统检测到异常后在物理主机上重建了虚拟机,重建的虚拟机与崩溃的虚拟机一样并保存了原虚拟机的永存数据;2)在虚拟机上打开多个任务窗口导致虚拟机宕机,系统检测到异常后对虚拟机进行了重启。虚拟机恢复时间如表 3 所示。

表 3　虚拟机恢复时间

故障原因	VM1	VM2	VM3
虚拟机被删	<40 s	<30 s	<1 min
虚拟机宕机	<30 s	<25 s	<50 s

(3) 虚拟应用故障:在虚拟机上运行有 FTP 服务器、在线视频库和 Web 服务器为用户提供 FTP 下载、在线视频观看和网站访问服务,在虚拟机上通过任务管理器将应用程序进程关闭,系统检测到进程异常后进行恢复,继续为用户提供服务。虚拟应用恢复时间如表 4 所示。

表 4　虚拟应用恢复时间

应用名称	VM1	VM2	VM3
FTP 下载	<30 s	<20 s	<1 min
在线视频	<50 s	<45 s	<2 min
Web 服务器	<1 min	<45 s	<2 min

通过实验结果表明,本文系统能够在虚拟化条件下自动检测云平台运行状态并获取相关参数,通过对数据分析判断发现异常,根据既定策略进行故障排除,完成虚拟机及应用的重启重建,恢复中断的服务,实现对云平台故障实时无代理恢复。

6　结束语

随着云计算应用日益广泛,云服务的可靠性要求也越来越高,云平台中的物理主机崩溃、虚拟机宕机、云应用无响应等故障都会导致云服务的中断,影响用户体验。本文利用 OpenStack 云平台设计实现的基于事件驱动机制的虚拟化故障检测恢复系统,通过对云平台运行参数的获取和分析处理,同时充分考虑负载均衡,实现了对云平台的实时检测和故障的无代理自动恢复,有效缩短了故障时间,提升了云平台的高可用性。由于本文方法并未考虑大数量故障的情况,数量增多会对故障的检测和恢复造成不利的影响,比如耗时较多、负载过大等,因此提高大数量故障检测效率将是下一步研究的重点。

(下转第 16 页)

5 结束语

本文提出一种基于改进蚁群算法的并行任务调度模型。将用户提交的动态任务分割成具有相互制约关系的子任务,并将其按运行次序放入具有不同优先级的调度队列中。采用基于最短任务延迟时间和任务分配公平的改进蚁群算法 DSFACO,对同一个调度队列中的子任务进行调度。DSFACO 算法以缩短任务延迟时间为目标研究子任务的调度问题。通过 Cloudsim 平台对比分析了 DSFACO 算法与 TS-EACO 算法,实验结果表明 DSFACO 算法综合性能优于 TS-EACO 算法,更加适应云计算环境。下一步工作的重点是在保证 DSFACO 算法具有最短任务延迟时间的同时降低总计算成本。

参考文献

[1] Armbmst M,Fox A,Griffith R,et al. Above the Clouds: A Berkeley View of Cloud Computing[R]. University of California, Berkeley, Technical Report: UCB/EECS-2009-28,2009.

[2] 祝家钰,肖 丹. 云环境下基于路径优先级的任务调度算法[J]. 计算机工程与设计, 2013, 34(10): 3511-3515.

[3] 孙 月,于 炯,朱建波. 云计算中一种多 DAG 工作流可抢占式调度策略[J]. 计算机科学,2014,41(3): 145-148.

[4] Dorigo M,Blum C. Ant Colony Optimization Theory: A Survey [J]. Theoretical Computer Science, 2005, 344(2/3):243-278.

[5] Gao Y. A Multi-objective Ant Colony System Algorithm for Virtual Machine Placement in Cloud Computing[J]. Journal of Computer and System Sciences,2013,79(8): 1230-1242.

[6] Dorigo M, Birattari M, Stutzel T. Ant Colony Optimization [J]. IEEE Computational Intelligence Magazine,2006,1(4):28-39.

[7] Huang Qiyi, Huang Tinglei. An Optimistic Job Scheduling Strategy Based on QoS for Cloud Computing[C]// Proceedings of 2010 IEEE International Conference on Intelligent Computing and Integrated Systems. [S. l.]:IEEE Press,2010:673-675.

[8] Sanyal M G. Survey and Analysis of Optimal Scheduling Strategies in Cloud Environment [C]//Proceedings of IEEE International Conference on Information and Communication Technologies. [S. l.]: IEEE Press, 2012:789-792.

[9] Chang F, Ren J, Viswanathan R. Optimal Resource Allocation in Clouds [C]//Proceedings of the 3rd International Conference on Cloud Computing. [S. l.]: IEEE Press,2010:418-425.

[10] 查华英,杨静丽. 改进蚁群算法在云计算任务调度中的应用[J]. 计算机工程与设计, 2013, 34(5): 1716-1726.

[11] 李建峰,彭 舰. 云计算环境下基于改进遗传算法的任务调度算法[J]. 计算机应用,2011,31(1):184-186.

[12] Dutta D,Joshi R C. A Genetic: Algorithm Approach to Cost-based Multi-QoS Job Scheduling in Cloud Computing Environment [C]//Proceedings of International Conference and Workshop on Emerging Trends in Technology. Mumbai, India: ACM Press, 2011: 422-427.

[13] 李 震,杜中军. 云计算环境下的改进型 Map-Reduce 模型[J]. 计算机工程,2012,38(11):27-29,37.

编辑 陆燕菲

(上接第 11 页)

参考文献

[1] Feng Dengguo,Zhang Min,Zhang Yan,et al. Study on Cloud Computing Security [J]. Journal of Software, 2011,22(1):71-83.

[2] Zissis D, Lekkas D. Addressing Cloud Computing Security Issues [J]. Future Generation Computer Systems,2012,28(3):583-592.

[3] Varia J. Cloud Architectures-Amazon Web Service[EB/OL]. (2009-03-01). http://acmbangalore, org/events/monthly-talk/may-2008-cloud-architectures-amazon-web-services. html.

[4] 林 闯,苏文博,孟 坤,等. 云计算安全:架构,机制与模型评价[J]. 计算机学报,2013,36(9):1765-1784.

[5] 陈海波,夏虞斌,陈 榕. 高可信、高扩展与高可用云计算平台的研究与展望[J]. 高性能计算发展与应用, 2013,43(2):29-34.

[6] Calzolari F, Arezzini S, Ciampa A, et al. High Availability Using Virtualization[J]. Journal of Physics, 2010,219(5).

[7] Tate J, Kelley R, Maliska S R R, et al. IBM SAN Solution Design Best Practices for VMware vSphere ESXi[Z]. IBM Redbooks,2013.

[8] OpenStack [EB/OL]. (2014-02-01). http://www. openstack. org.

[9] Litvinski O, Gherbi A. Experimental Evaluation of OpenStack Compute Scheduler[J]. Procedia Computer Science,2013,19(1):116-123.

[10] Sotomayor B,Montero R S,Llorente I M,et al. Virtual Infrastructure Management in Private and Hybrid Clouds[J]. IEEE Internet Computing, 2009, 13(5): 14-22.

[11] 王 莉,李新明,李 艺,等. 高可用性系统软件 HHA 的事件驱动机制[J]. 计算机工程与应用, 2003, 39(4):145-147.

[12] Pimentel J R. An Incremental Approach to Task and Message Scheduling for Autosar Based Distributed Automotive Applications[C]//Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems. [S. 1.]:IEEE Computer Society, 2007:1-7.

[13] Wen X,Gu G,Li Q,et al. Comparison of Open-source Cloud Management Platforms: OpenStack and OpenNebula[C]//Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery. [S. 1.]:IEEE Press,2012:2457-2461.

编辑 索书志