基于粒子群优化与蚁群优化的云计算任务调度算法

王登科 李 忠
(西北师范大学数学与信息科学学院 甘肃 兰州 730070)

摘 要 在云计算环境中用户数量众多,系统要处理的任务量十分巨大,为了使系统能够高效地完成服务请求,如何对任务进行调度成为云计算研究的重点。提出一种基于粒子群优化和蚁群优化的任务调度算法,该算法首先利用粒子群优化算法迅速求得初始解,然后根据该调度结果生成蚁群算法的初始信息素分布,最后利用蚁群算法得到任务调度的最优解。通过在 CloudSim 平台进行仿真实验,表明该算法具有较好的实时性和寻优能力,是一种有效的调度算法。

关键词 云计算 任务调度 粒子群优化 蚁群优化

中图分类号 TP311 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2013.01.075

A TASK SCHEDULING ALGORITHM BASED ON PSO AND ACO FOR CLOUD COMPUTING

Wang Dengke Li Zhong
(*College of Mathematics and Information Science, Northwest Normal University, Lanzhou 730070, Gansu, China*)

Abstract In cloud computing environment, there are a large number of users which lead to huge amount of tasks to be processed by system. In order to make the system complete the service requests efficiently, how to schedule the tasks becomes the focus of cloud computing research. A task scheduling algorithm based on PSO and ACO for cloud computing is presented in this paper. First, the algorithm uses particle swarm optimisation algorithm to get the initial solution quickly, and then according to this scheduling result the initial pheromone distribution of ant colony algorithm is generated. Finally, the ant colony algorithm is used to get the optimal solution of task scheduling. The experiment simulated on CloudSim platform shows that the algorithm has good effect in real-time performance and optimisation capability. It is an effective task scheduling algorithm.

Keywords Cloud computing Task scheduling Particle swarm optimisation (PSO) Ant colony optimisation(ACO)

0 引 言

云计算是分布式计算、并行计算、网格计算、效用计算、虚拟化、网络存储等传统计算机和网络技术发展融合的商业实现。它是一种基于互联网的计算服务模式,它实现了对共享可配置计算资源(网络、服务器、存储、应用和服务等)的方便、按需访问;这些资源可以通过极小的管理代价或者与服务提供者的交互被快速地准备和实现按需使用。云服务可分为 3 层:基础设施即服务(IaaS),平台即服务(PaaS)和软件即服务(SaaS)[1]。云计算的基本原理是,通过网络将庞大的计算处理程序自动分拆成许多较小的子程序,再由多部服务器所组成的庞大系统搜寻、计算、分析之后将处理结果回传给使用者。通过这项技术,远端的服务供应商可以在数秒之内处理 TB 级的数据量,达到与超级计算机同样强大效能的网络服务[2]。如 Google 提出的 Map/Reduce 框架。

在商业领域,除了数据挖掘、搜索、数据存储可以采用离线任务的模式外,其他的所有应用(如电子商务,交易处理等)都是对实时性要求较高的业务,这就要求云计算系统能够在尽可能短的时间内完成用户提交的任务。由于在云环境中,用户数量众多,系统要处理的任务量十分巨大,因此为了使系统能够快速处理服务请求,生成服务实例,返回服务数据,在“云”中如何对任务进行高效合理的调度,实现系统全局最优化,成为云计算研究的重点与难点[3]。

在云计算环境中,任务调度是一个 NP 完全问题[4]。模拟退火、遗传、蚁群、粒子群等智能优化算法非常适合于求解 NP 类问题[5]。遗传算法、粒子群优化算法、蚁群算法已被用于求解云计算的任务调度问题。文献[6]运用遗传算法,在云计算环境下对独立任务进行了有效的调度。文献[7]提出了一种基于粒子群优化的云计算任务调度算法,优化了任务的传输时间和处理时间。文献[8]利用蚁群算法对云计算任务进行调度,使总的任务完成时间和平均任务完成时间达到最小。但是任何算法都存在其优势与不足,例如遗传算法虽然具有快速随机的全局搜索能力,但参数较多,编程实现比较复杂,且容易陷入局部最优;粒子群优化算法初期收敛速度快,后期局部搜索能力不足,收敛速度缓慢,但同遗传算法相比,粒子群优化算法收敛速度快,优化性能上好于遗传算法[9],并且编程易于实现,需要调整的参数少;而蚁群算法虽具有较好的寻优能力,但初期信息素匮乏,收敛速度慢。在此研究的基础上,本文提出一种基于粒子

收稿日期:2012-06-01。王登科,硕士生,主研领域:云计算,企业信息化。李忠,教授。

群优化与蚁群优化的云计算任务调度算法。该算法吸收了粒子群优化算法快速收敛和蚁群算法的寻优能力,缩短了系统处理调度问题的时间,减少了总的任务执行时间,提高了云计算任务调度的效率。

1　云计算任务调度的描述

目前,云计算环境大部分采用 Google 提出 Map/Reduce 的编程模式[10],通过 Map(映射)和 Reduce(化简)两个阶段,将一个较大的任务分割成为许多较小的子任务,然后分配给若干个虚拟资源节点并行执行,最后返回运行结果。本文只考虑子任务相互独立的情况。云计算任务调度的目标就是通过对各子任务进行合理的资源分配,从而使得总的任务完成时间达到最小。

云计算任务调度问题可描述为: 在云环境中,将 n 个相互独立的子任务分配给 m 个虚拟资源节点执行($m < n$) 。其中,任务集表示为 $T = \{t_1, t_2, \cdots, t_n\}$,虚拟资源节点集表示为 $VM = \{vm_1, vm_2, \cdots, vm_m\}$,$t_j(j = 1,2,\cdots,n)$ 表示第 j 个子任务,$vm_i(i = 1,2,\cdots,m)$ 表示第 i 个虚拟资源。每个子任务只能在一个虚拟资源上执行。任务集 $T = \{t_1, t_2, \cdots, t_n\}$ 与虚拟资源节点集 $VM = \{vm_1, vm_2, \cdots vm_m\}$ 的分配关系可用矩阵 X 表示为:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix} \tag{1}$$

其中,x_{ij} 表示子任务 t_j 与虚拟资源 vm_i 的分配关系,$x_{ij} \in \{0,1\}$,$\sum_{i=1}^{m} x_{ij} = 1$,$i \in \{1,2,\cdots,m\}$,$j \in \{1,2,\cdots,n\}$,如果子任务 t_j 在资源 vm_i 上执行,则 $x_{ij} = 1$,否则 $x_{ij} = 0$。ETC_{ij} 为子任务 t_j 在虚拟资源 vm_i 的期望执行时间。对应任务与虚拟资源的分配关系矩阵 X,可以构成 ETC 矩阵:

$$ETC = \begin{pmatrix} ETC_{11} & ETC_{12} & \cdots & ETC_{1n} \\ ETC_{21} & ETC_{22} & \cdots & ETC_{2n} \\ \cdots & \cdots & & \cdots \\ ETC_{m1} & ETC_{m2} & \cdots & ETC_{mn} \end{pmatrix} \tag{2}$$

b_j 为任务 t_j 执行的起始时间。$CT_{ij}(i \in \{1,2,\cdots,m\}, j \in \{1,2,\cdots,n\})$ 为 t_j 在虚拟资源 vm_i 执行的期望完成时间,$CT_{ij} = b_j + ETC_{ij}$。$\max\{CT_{ij}\}$ 为总任务的完成时间,$C_{\max} = \max\{CT_{ij}\}$ 为该调度的适应值。优化目标是使 $C_{\max}$ 的值最小。

2　云计算任务调度的设计

粒子群优化算法[11] 是由 Kennedy 和 Eberhart 受鸟群觅食行为的启发,在 1995 年提出的一种仿生优化算法,由于算法简单,易于实现 ,参数少等特点在连续优化问题和离散优化问题中表现出良好的效果。蚁群算法[12] 是由 Dorigo Macro 等人在 20 世纪 90 年代提出的一种模拟蚂蚁群体觅食行为的仿生优化算法。该算法引入正反馈并行机制,具有较强的鲁棒性、优良的分布式计算机制、易于与其他算法相结合等优点,被广泛应用于解决各种复杂组合优化问题。

虽然粒子群优化算法和蚁群算法在解决优化问题时分别取得了较好的效果,但是每种算法都存在其不足之处。粒子群优化算法在初始阶段收敛速度快,但在后期由于种群缺乏多样性,局部搜索能力不足,收敛速度缓慢,而蚁群算法虽然具有较好的寻优能力,但由于初始阶段信息素匮乏,初期收敛速度较慢。本文结合两种算法的优势,提出一种基于粒子群优化与蚁群优化的云计算任务调度算法。该算法首先利用粒子群优化算法的快速收敛性迅速生成初始解,然后根据该调度结果生成蚁群算法的初始信息素分布[13],最后利用蚁群算法求得任务调度的最优解。

2.1　粒子群算法

云计算任务调度问题是离散问题,而粒子群优化算法主要适于解决连续优化问题。在离散版的粒子群算法中,粒子位置向量的每一位取值为 0 或 1,粒子速度不再表示连续空间中粒子飞行的速度,其意义在于计算粒子位置向量取值为 0 或 1 的概率。

(1) 粒子位置与速度的定义

根据云计算任务调度的特点,需要对粒子位置和速度进行重新定义。在粒子群中,每个粒子的位置代表一个可行的任务调度方案,根据分配关系矩阵 X,第 i 个粒子的位置可表示为:

$$X_i = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix} \tag{3}$$

其中,$x_{ij} \in \{0,1\}$,$\sum_{i=1}^{m} x_{ij} = 1$,$i \in \{1,2,\cdots,m\}$,$j \in \{1,2,\cdots,n\}$。

类似的,粒子 i 的速度定义为:

$$V_i = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{pmatrix} \tag{4}$$

其中,$v_{ij} \in [-v_{\max}, v_{\max}]$,$i \in \{1,2,\cdots,m\}$,$j \in \{1,2,\cdots,n\}$。

(2) 粒子位置与速度的更新

在离散粒子群算法中,为了使粒子的位置向量 x_{ij}^{k+1} 取值为 0 或 1,引入模糊函数 $Sig(v_{ij}^{k+1})$,其公式为:

$$sig(v_{ij}^{k+1}) = \frac{1}{1 + \exp(-v_{ij}^{k+1})} \tag{5}$$

粒子速度与位置的更新公式为:

$$v_{ij}^{k+1} = v_{ij}^{k} + c_1 r_1 (p_{ij}^{k} - x_{ij}^{k}) + c_2 r_2 (p_{gj}^{k} - x_{ij}^{k}) \tag{6}$$

$$x_{ij}^{k+1} = \begin{cases} 1 & r_3 < Sig(v_{id}^{k+1}) \\ 0 & \text{else} \end{cases} \tag{7}$$

其中,k 表示迭代次数,p_{ij} 为粒子个体最优位置的 j 维向量,p_{gd} 为粒子群体最优位置的 j 维向量,c_1 和 c_2 为学习因子,通常 $c_1 = c_2 = 2$,$r_1, r_2, r_3 \in U[0,1]$。速度 v_{ij}^{k+1} 被限定在 $[-v_{\max}, v_{\max}]$ 内,当 $v_{ij}^{k+1} < -v_{\max}$ 时,令 $v_{ij}^{k+1} = -v_{\max}$,当 $v_{ij}^{k+1} > v_{\max}$ 时,令 $v_{ij}^{k+1} = v_{\max}$。

2.2　蚁群算法

参照云计算任务调度的任务与资源分配关系矩阵 X,定义 $\{x_{ij}\}_{m \times n}$ 为节点集,组成一个无向完全图 $G(V,E)$。通过蚁群算法从集合 $\{x_{ij}\}_{m \times n}$ 中选出一组节点 $\{x_{y_1 1}, x_{y_2 2}, \cdots, x_{y_j j}, \cdots, x_{y_n n}\}$,$y_j \in \{1,2,\cdots,m\}$ 使得 $C_{\max}$ 的值最小。

(1) 信息素初始化

在蚁群算法的初始化阶段,根据粒子群算法得到的调度结果,将初始解节点上的信息素加强一定倍数,使得蚁群算法的加速收敛,让蚂蚁在游历过程中能够很快接近最优解。通过多次实验发现信息素加强 6 倍时效果较好。

(2) 路径的选择

将若干只蚂蚁随机放置于节点上进行周游，第 k 只蚂蚁在 t 时刻选择节点 x_{ij} 的转移概率为:

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{x_{is}\notin tabu_k}[\tau_{is}(t)]^{\alpha}[\eta_{is}(t)]^{\beta}} & x_{ij}\notin tabu_k\\ 0 & \text{else}\end{cases} \tag{8}$$

$$\eta_{ij}(t)=\frac{1}{CT_{ij}^{k}(t)} \tag{9}$$

$\tau_{ij}(t)$ 表示 t 时刻节点 x_{ij} 上残留的信息素。$\eta_{ij}(t)$ 是启发信息。$tabu_k(k=1,2,\cdots,m)$ 为蚂蚁 k 的禁忌表，每个子任务 t_j 对应一组节点 $\{x_{ij}\}_{i=1}^{m}$，当 t 时刻蚂蚁 k 选择节点 x_{ij} 后，把 $\{x_{ij}\}_{i=1}^{m}$ 加入禁忌表 $tabu_k$。α 和 β 分别表示信息量与启发信息的相对重要程度。

(3) 信息素的更新

当蚂蚁选择一个节点后，即当任务被分配到某个虚拟资源时，节点的信息素会发生变化，根据式(9)、式(10)对信息素进行局部更新:

$$\tau_{ij}(t+1)=(1-\rho)\cdot\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \tag{10}$$

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}\dfrac{Q}{CT_{ij}^{k}(t)} & \text{第 } k \text{ 只蚂蚁在 } t \text{ 和 } t+1 \text{ 之间选择 } x_{ij}\\ 0 & \text{else}\end{cases} \tag{11}$$

其中，$\rho\in[0,1)$ 是信息素挥发系数，$1-\rho$ 表示信息的残留系数。$\Delta\tau_{ij}^{k}(t)$ 表示第 k 只蚂蚁在本次循环中留在路径 (i,j) 上的信息量。Q 为常量。

当蚂蚁完成一次循环后，即当所有任务完成调度后，根据式(12)和式(13)对信息素进行全局更新:

$$\tau_{ij}(t+n)=(1-\rho)\cdot\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \tag{12}$$

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}\dfrac{Q}{C_{\max}^{k}} & \text{第 } k \text{ 只蚂蚁本次循环中选择 } x_{ij}\\ 0 & \text{其他}\end{cases} \tag{13}$$

2.3 算法流程

基于粒子群优化与蚁群优化的云计算任务调度算法的基本执行步骤如下:

Step1 定义云计算任务调度的目标函数。

Step2 设定该调度算法的相关参数和算法结束条件。

Step3 对粒子群的位置和速度进行随机初始化。

Step4 计算每个粒子的适应值，找出粒子个体最优位置和群体最优位置。

Step5 根据式(6)、式(7)对粒子速度和位置进行更新。

Step6 无动作。

Step7 如果达到设定的迭代次数，输出云计算任务调度的初始调度结果，转至 Step8。否则，跳转至 Step4。

Step8 根据粒子群算法得出的初始调度结果对蚁群算法的信息素进行初始化。

Step9 将若干只蚂蚁随机放置于节点上进行搜索。

Step10 每只蚂蚁根据式(8)选择选择下一节点，并根据式(10)、式(11)更新局部信息素，并把所选节点加入到调度列表。

Step11 当所有任务完成调度后，根据任务调度列表计算调度结果的适应值，并根据式(12)、式(13)更新全局信息素。否则，跳转至 Step10。

Step12 蚁群算法达到最大迭代次数，算法结束，输出云计算任务调度的最优解。否则，跳转至 Step9。

3 实验与分析

CloudSim[14] 是澳大利亚墨尔本大学 Rajkumar Buyya 博士领导团队开发的云计算仿真平台，主要用于模拟云环境，对不同应用和服务模型的调度策略进行性能测试。CloudSim 中的 DataCenterBroker 类用于模拟一个代理，负责根据服务质量需求在线协商任务与资源的分配策略。用户可以通过在 DataCenterBroker 类中添加一个实现自定义调度算法的方法来对该类进行扩展，从而完成对调度算法的模拟。

本文采用 CloudSim 平台来模拟云环境，对任务调度算法进行模拟仿真。为了检验本文调度算法的性能，分别在 5 个虚拟计算资源、50 个子任务和 5 个虚拟计算资源、500 个子任务两种调度规模下，用粒子群优化算法(PSO)、蚁群算法(ACO)和本文算法(PSO-ACO)进行了对比实验。实验中各虚拟计算资源的处理能力为{400MIPs,600MIPs,800MIPs,1000MIPs,1200MIPs}，子任务的任务长度范围为[400MI,1000MI]。各算法的参数设置如下: 本文算法的粒子群算法部分，群体规模 $size=100$，$c_1=c_2=2$，迭代次数为 40，蚁群算法部分，群体规模 $size=100$，$\alpha=1$，$\beta=1$，$\rho=0.2$，迭代次数为 160。粒子群算法、蚁群算法的参数与本文算法粒子算法部分、蚁群算法部分设置的参数相同，迭代次数为 200。各算法重复运行 20 次。实验结果如图 1 和图 2 所示。

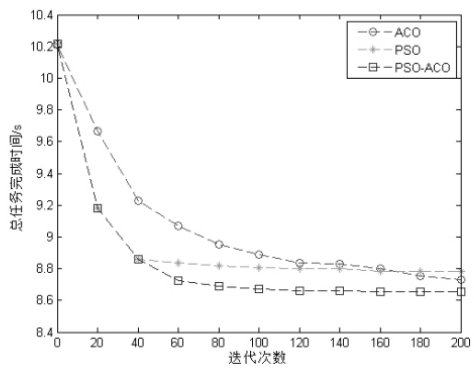


图 1 总任务完成时间曲线(5 个资源，50 个子任务)

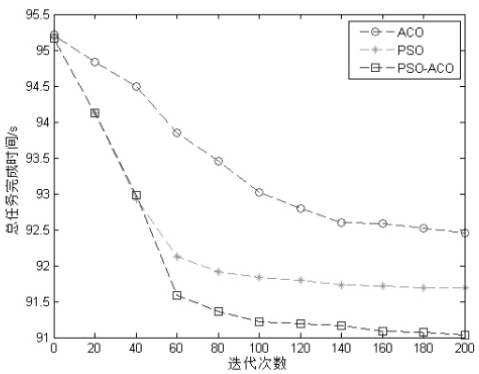


图 2 总任务完成时间曲线(5 个资源，500 个子任务)

从上面的实验结果可以看出，本文所提的 PSO-ACO 任务调度算法结合了 PSO 的快速收敛能力和 ACO 的寻优能力，在迭代初期，采用 PSO-ACO 对任务进行调度的优化效果好于 ACO，其

总任务完成时间小于 ACO 调度的总任务完成时间，在迭代后期，PSO-ACO 的调度效果优于 PSO 和 ACO，能够得到最优解，总任务完成时间最小。同时可以看到，PSO-ACO 能够在相对较小的代数内找到比 PSO 和 ACO 较好的调度结果。从图中可以看出，采用 PSO 的调度优化效果，迭代初期比较明显，总任务完成时间比 ACO 的小，但是随着迭代次数的增加，其优化效果变弱，总任务完成曲线趋于平缓。这是由于 PSO 具有初期收敛速度快，后期局部搜索能力不足的特点所造成的。在这两个实验结果中，采用 ACO 调度的优化效果差别较大，在小规模调度情况下，ACO 在初期的总任务完成时间大于 PSO-ACO 和 PSO 的总任务调度时间，在后期 ACO 的调度结果优于 ACO，总任务时间小于 PSO; 大规模调度情况下，ACO 在整个迭代过程中，调度效果较差，总任务完成时间大于 PSO-ACO 和 ACO 的任务完成时间。这是由于在 ACO 虽然具有较好的寻优能力，但初始阶段信息素匮乏，初期收敛速度较慢，同时蚁群算法还会随着问题规模的增大，收敛速度变慢，易陷入局部最优。

通过在两种调度规模下进行仿真实验，结果表明本文所提的调度算法缩短了系统处理调度问题的时间，减少了总任务的完成时间，在时间性能和优化性能上取得了较好的效果。

4　结　语

本文针对云计算任务调度问题，提出了一种基于粒子群优化与蚁群优化的云计算任务调度算法。通过仿真实验，证明了该算法具有较好的实时性和寻优能力，是一种有效的调度算法。本文算法只考虑了任务调度的时间因素，而在实际应用中情况比较复杂，还有更多因素需要考虑，比如计算成本、负载均衡、节能，因此还有待作进一步的研究，以使对云计算任务的调度在综合性能上达到最优。

参　考　文　献

[1] 朱近之. 智慧的云计算 [M]. 北京: 电子工业出版社，2010.

[2] Wikipedia. Cloud computing [EB/OL]. [2012-05-21]. http://de.wikiped ia.org/wiki/Cloud_Computing.

[3] 陈全，邓倩妮. 云计算及其关键技术 [J]. 计算机应用，2009，9(29): 2562-2567.

[4] Arfeen M A, Pawlikowski K, Willig A. A Framework for Resource Allocation Strategies in Cloud Computing Environment [J]. Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual，2011: 261-266.

[5] 汪定伟，王俊伟，王洪峰，等. 智能优化方法 [M]. 北京: 高等教育出版社，2007.

[6] Zhao Chenhong, Zhang Shanshan, Liu Qingfeng, et al. Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing [C]//Proc IEEE 5th International Conference on Wireless Communications, Networking and Mobile Computing WiCom'09, Beijing，2009: 1-4.

[7] Guo Lizheng，Zhao Shuguang，Shen Shigen，et al. Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm [J]. Journal of Networks. 2012，7(3): 547-553.

[8] Li Jianfeng, Peng Jian, Cao Xiaoyang，et al. A Task Scheduling Algorithm Based on Improved Ant Colony Optimization in Cloud Computing Environment [J]. Energy Procedia, 2011(13): 6833-6840.

[9] Kennedy J, Spears W. Matching Algorithms to Problems: an Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator [C]//Proc IEEE International Conference on Evolutionary Computation. Piscataway, NJ: IEEE Service Center，1998: 78-83.

[10] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C]//Proceedings of the 6th Symposium on Operating System Design and Implementation. New York: ACM, 2004: 137-150.

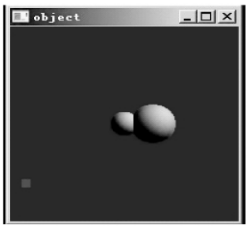
[11] Kennedy J, Eberhart R C. Particleswarm ptimization [C]//Proc IEEE International Conference on Neural Networks, IV. Piscataway, NJ: IEEE Service Center, 1995: 1942-1948.

[12] Dorigo M, Maniezzo V, Colorni A. The ant system: optimization by a colony of cooperating agent [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1996, 26(1): 29-41.

[13] 丁建立，陈增强，袁著祉. 遗传算法与蚂蚁算法的融合 [J]. 计算机研究与发展，2003，9(40): 1351-1356.

[14] Calheiros R N，Ranjan R, Rose C A F D, et al. CloudSim: a novel framework for modeling and simulation of cloud computing infrastructures and services [EB/OL]. [2009-9-3]. http://arxiv.org/abs/0903.2525.

(上接第 272 页)



(c) 碰撞

图 5　仿真实验

6　结　语

本文通过采用离屏渲染技术 FBO 和 GPGPU，将物体深度数据渲染至纹理，然后对纹理中存储的物体最大、最小深度值进行比较运算，以确定两物体是否碰撞，从而充分利用了图形硬件 GPU 的运算性能。从仿真结果可以看出，该方法能正确地返回碰撞检测结果。

参　考　文　献

[1] Vaneeek G，Jr. BreP-index. A Multidimensional Space PartitioningTree [J]. Comput. Gecom. Appl 1991，1(3): 243-261.

[2] 范昭炜. 实时碰撞检测技术研究 [D]. 杭州: 浙江大学，2003.

[3] Jang Han-Young，Jeong Taek Sang，Han Jung Hyun. Image-Space Collision Detection Through Alternate Surface Peeling [C]//ISVC. USA: Springer，2007(1): 66-75.

[4] 范昭炜，万华根，高曙明. 基于图像的快速碰撞检测算法 [J]. 计算机辅助设计与图形学学报，2002，14(9): 805-810.

[5] 霍滨焱. 基于图像空间的碰撞检测算法 [D]. 哈尔滨: 哈尔滨工程大学，2005.

[6] 宋永军，苏鸿根. 一种基于图像的刚体碰撞检测 [J]. 计算机应用与软件，2004，21(5): 82-84.

[7] 王季，翟正军，蔡小斌. 基于深度纹理的实时碰撞检测算法 [J]. 计算机辅助设计与图形学学报，2007，19(1): 59-63，68.

[8] Ezio Todini. Influence of parameter estimation uncertainty in Kriging part—theoretical development [J]. Hydrology and Earth System Sciences，2001，5(2): 215-223.