

基于云环境下的负载均衡算法研究

张水平¹, 庞 敏², 仲伟彪², 何迎盛²

(1 江西理工大学 现代教育技术及信息中心, 江西 赣州 341000; 2 江西理工大学 信息工程学院, 江西 赣州 341000)

摘 要: 针对传统遗传算法易陷入局部解等缺点, 提出了一种改进的元胞遗传算法, 并将其运用到云环境下的负载均衡中。在云环境中, 由于用户任务请求量大, 会出现有些服务器过载, 而有些服务器闲散的问题, 利用该算法找到能提高服务器利用率的方法。在 Clousim3.0 仿真平台上进行试验, 结果表明, 该算法能提高服务器利用率, 缩短任务完成时间。

关键词: 云计算; 负载均衡; 遗传算法; 元胞自动机

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2014)08-0049-04

Research on Load Balancing Based on Cloud Computing

ZHANG Shui-ping¹, PANG Min², ZHONG Wei-biao², HE Ying-sheng²

(1 Modern Educational Technology and Information Center, Jiangxi University of Science and Technology, Ganzhou 341000, China; 2 Faculty of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China)

Abstract: To the problem of traditional genetic Algorithm to local solution, this paper proposes a improved cellular genetic algorithm and applys to the load balancing under cloud computing. Under the cloud environment, because of the large amount of request, some servers get overloaded, but others are free, takes use of this algorithm to search for a way to improve servers utilization. Taking the experiment under the Clousim3.0, the results shows that this algorithm can improve servers utilization and reduce the whole makespan.

Key words: cloud computing; load balancing; genetic algorithm; cellular automata

1 引言

云计算的核心思想是^[1], 统一管理和调度网络连接的计算资源, 构成一个计算资源池, 根据用户的需要提供服务。但是当网络的访问量和数据量快速增长, 其处理能力和计算强度也相应地增大, 面对应用或工作负载的突发状况, 单一的服务器节点会出现热点问题, 无法承担突发的 workflow。目前针对这个问题仍没有非常好的解决方法。遗传算法^[2-3]作为一种智能算法, 它有内在的隐并行性和更好的全局寻优能力等优点, 但它仍存在一些缺陷, 如搜索后期效率低、早熟、陷入局部解等。本文在传统遗传算法的基础上, 结合元胞自动机, 将元胞遗传算法应用于云环境下的负载均衡中。

2 元胞自动机及云环境下负载模型描述

2.1 元胞自动机简介

元胞自动机(Cellular Automaton, CA)^[4], 是一时间和空间都离散的动力系统。每个元胞取有限的离散状态, 分布在规则网格中, 根据确定的局部规则作同步更新。将元胞自动机每个时刻状态的变化看成是对信息或数据的处理或计算时, 则元胞自动机在进行同步处理, 特别适合于并行计算。

元胞自动机最基本的组成包括元胞、邻居、元胞空间及规则四部分。用数学符号来表示, 标准的元胞自动机是一个四元组: $A = (L_d, S, N, f)$ 。

A 代表一个元胞自动机系统。 L 代表元胞空间, d 是一正整数, 表示元胞自动机内元胞空间的维数。

本文取 $d = 2$. S 是元胞在每个时刻的状态. 每个元胞, 在每个时刻只有一个状态, 如 1 和 0, 它们各自的状态随时间变化. N 表示一个所有领域内元胞的组合. 记为 $N = (S_1, S_2, \dots, S_n)$, n 是元胞邻居个数. 常见的有三种元胞邻居模型, 如图 1 所示. 冯 - 诺依曼型有 4 个邻居; 摩尔型有 8 个邻居; 扩展摩尔型有 24 个邻居. 本文选择摩尔型.

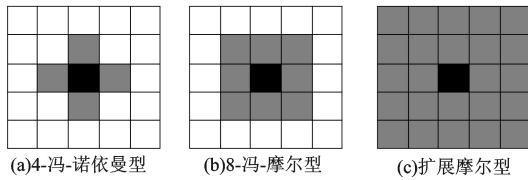


图 1 元胞邻居模型

f 是演化规则. 根据元胞和其邻居元胞的当前状态以及演化规则来判断元胞下一时刻的状态, 记为 $s_i^{t+1} = f(s_i^t, s_n^t)$, 演化规则是元胞自动机的核心.

2.2 云环境下的负载模型

在云计算环境中, 负载均衡涵盖两种不同的意义, 其中第一级负载体现在负责执行任务的虚拟机层, 第二级负载体现在负责为虚拟机提供资源的物理主机层, 两级之间形成链式负载关系^[5], 图 2 中显示了云环境下服务器的负载模型. 本文从一级负载去研究云环境下的负载均衡.

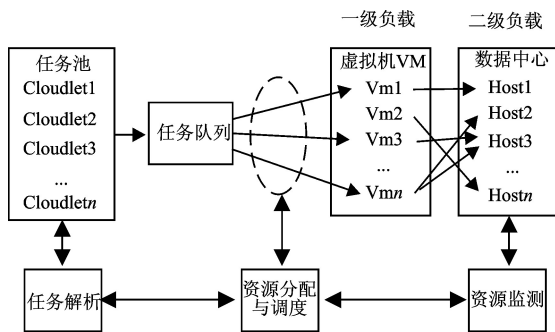


图 2 云环境下两级链式负载模型

3 适应度函数建立

负载均衡有两方面的含义: 第一, 为了减少用户等待响应的时间, 大量的数据流量或并发访问分担到多台节点设备上分别处理; 第二, 单个重负载的运算分担到多台节点设备上做并行处理, 每个节点设备处理完毕后, 将结果汇总, 返回给用户^[6]. 所以, 本文从任务响应时间和服务器的利用率两方面考虑, 设计适应度函数.

$Q[q_1, q_2, \dots, q_n]$ 表示用户任务集合, 其中, q_i ($i = 1, 2, \dots, n$) 表示第 i 个任务, 描述为

$q_i \{q_{i,cpu}, q_{i,mem}\}$, 其中, $q_{i,cpu}$ 表示第 i 个任务对处理器的要求, $q_{i,mem}$ 表示第 i 个任务对内存的要求.

$VM[v_{m_1}, v_{m_2}, \dots, v_{m_m}]$ 表示虚拟机集合, 其中, v_{m_j} ($j = 1, 2, \dots, m$) 表示第 j 个虚拟机, 描述为 $v_{m_j} \{v_{m_j,max_cpu}, v_{m_j,max_mem}\}$, 其中, v_{m_j,max_cpu} 表示第 j 个虚拟机的最大处理能力, v_{m_j,max_mem} 表示第 j 个虚拟机的最大内存.

$x(m, n)$ 表示分配方案矩阵, 元素 $x_{j,i} = \{0, 1\}$, 其中, $j = 1, \dots, m, i = 1, \dots, n$, 当 $x_{j,i} = 1$ 时, 表示第 i 个任务被分配到第 j 个虚拟机上.

假设虚拟机 v_{m_j} 已用 CPU 及内存为 $v_{m_j,used_cpu}$, $v_{m_j,used_mem}$, 则将任务分配到虚拟机 v_{m_j} 上后已用 CPU 及内存为

$$\begin{cases} v_{m_j,sumused_cpu} = v_{m_j,used_cpu} + \sum_{i=1}^n x_{j,i} * q_{i,cpu} \\ v_{m_j,sumused_mem} = v_{m_j,used_mem} + \sum_{i=1}^n x_{j,i} * q_{i,mem} \end{cases} \quad (1)$$

同时满足以下约束条件:

$$\begin{cases} v_{m_j,sumused_cpu} \leq v_{m_j,max_cpu} \\ v_{m_j,sumused_mem} \leq v_{m_j,max_mem} \end{cases} \quad (2)$$

虚拟机 v_{m_j} 的 CPU 与内存负载率为

$$\begin{cases} Load_{j,cpu} = \frac{v_{m_j,sumused_cpu}}{v_{m_j,max_cpu}} * 100\% \\ Load_{j,mem} = \frac{v_{m_j,sumused_mem}}{v_{m_j,max_mem}} * 100\% \end{cases} \quad (3)$$

则虚拟机 v_{m_j} 的总负载为

$$Load_{j,sum} = w * Load_{j,cpu} + v * Load_{j,mem} \quad (4)$$

其中, $w + v = 1$.

则所有虚拟机的平均负载为

$$Load_{average} = \frac{\sum_{j=1}^m Load_{j,sum}}{m} \quad (5)$$

虚拟机 v_{m_j} 与平均负载差为

$$|Load_{j,sum} - Load_{average}| \quad (6)$$

用二维数组统计每个任务运行在每个虚拟机上预计完成所需时间, 表示为

$$ETC[m][n] = \begin{bmatrix} ETC_{11} & ETC_{12} & \dots & ETC_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ ETC_{m1} & ETC_{m2} & \dots & ETC_{mn} \end{bmatrix} \quad (7)$$

式中, ETC_{nm} 表示第 n 个任务在第 m 个虚拟机上预计完成所需时间, 则第 k 个染色体上所有任务执行在对应虚拟机上预计完成时间为

$$\sum_{i=1}^n ETC_{ji}^k \quad (8)$$

式中, k 表示第 k 个染色体, j 的值对应于第 k 个染色体上的每个基因值。

综上所述, 适应度函数为

$$f(k) = k_1 \frac{1}{\sum_{j=1}^m |Load_{j, sum} - Load_{average}|} + k_2 \frac{1}{\sum_{i=1}^n ETC_{ji}^k} \quad (9)$$

式中, $k_1 + k_2 = 1$, k_1, k_2 分别表示负载率与任务完成时间在适应值中占的权重。

4 元胞遗传算法的改进

元胞遗传算法^[7]是将元胞自动机模型与遗传算法相结合而产生的, 是遗传算法的一种, 种群中的个体只与其邻居相作用。本文将元胞自动机应用到元胞的选择与交叉上。

(1) 选择

将每个个体的适应度值存放在一维数组中, 根据该数组建立二维元胞自动机的四方网格, 将各适应值填入四方网格中, 每个网格代表一元胞。本文采用摩尔型元胞, 将每个元胞与其 8 个邻居元胞进行适应度值比较, 将最大值作为该元胞的适应度值, 本文采用周期型边界, 重复上述操作, 直到满足循环次数。

(2) 交叉

在交叉过程中, 会出现两父代个体差异小, 交叉不产生新个体的情况。本文根据相关性的概念, 使相关性大的个体被选的概率小。若用 $p\left(\frac{Y_i}{X}\right)$ 表示 X 为中心元胞, 邻居元胞 Y_i 被选的概率时, 则有

$$p\left(\frac{Y_i}{X}\right) = 1 - \frac{1}{8} \left[1 + \lambda \frac{r(X, Y_i) - r_{avg}}{r_{max} - r_{min}} \right] \quad (10)$$

其中

$$r_{avg} = \frac{1}{8} \sum_{i=1}^8 r(X, Y_i) \quad i = 1, 2, \dots, 8 \quad (11)$$

$$r_{max} = \max\{r(X, Y_i)\} \quad i = 1, 2, \dots, 8 \quad (12)$$

$$r_{min} = \min\{r(X, Y_i)\} \quad i = 1, 2, \dots, 8 \quad (13)$$

式中, $r(X, Y_i)$ 表示中心元胞 X 对应的个体与邻居元胞 Y_i 对应的个体的相关性, 即两个个体中基因相同的个数, r_{avg} 表示中心元胞 X 对应的个体与所有邻居元胞 Y_i 对应的个体的相关性的平均值, r_{max} 表示中心元胞 X 对应的个体与邻居元胞 Y_i 对应的个

体相关性的最大值, r_{min} 表示中心元胞 X 对应的个体与邻居元胞 Y_i 对应的个体相关性的最小值, λ 为常数, 且有 $0 \leq \lambda \leq 1$ 。

在实际操作中, 将每个个体对应的索引值填入到二维元胞自动机的四方网格中, 选择一元胞, 根据其对应个体与邻居元胞对应个体的相关性, 得到每个元胞对应个体被选的概率, 最后通过轮盘赌的方法确定最终要进行交叉的父代个体。本文选择的交叉概率为 0.6, 产生随机交叉点, 实现父代个体的交叉。在交叉数的范围内, 重复上述过程。

(3) 变异

交叉操作完成后, 对产生的个体进行变异, 其中变异数目为变异范围(种群数 * 个体长度)与变异率的乘积。根据初始种群中染色体的生成方法, 在变异数目的范围内, 产生新个体。本文取变异率为 0.01。算法流程图如图 3 所示。

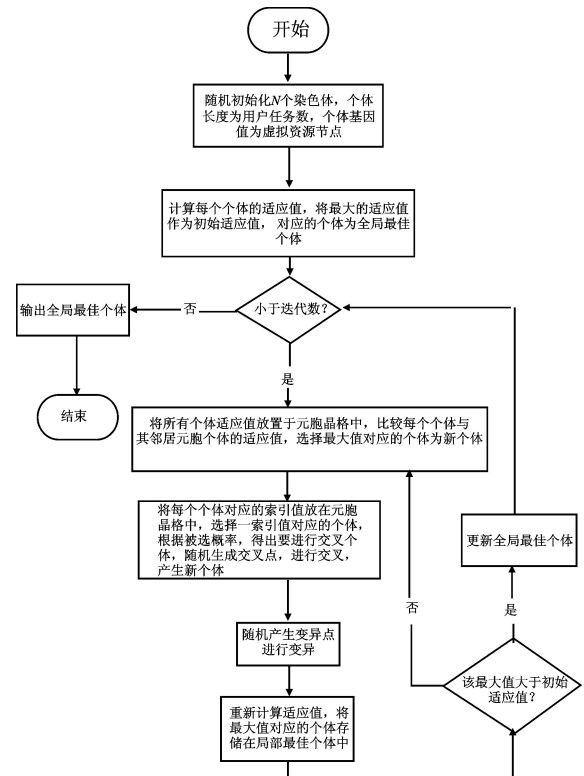


图 3 算法流程图

5 仿真实验与分析

本文基于最新版的 CloudSim3.0 云仿真平台^[8]进行仿真实验, 通过模拟云计算环境, 扩展 DataCenter Broker 类中的 bindCloudletToVM() 方法, 实现了对改进的负载均衡算法的模拟仿真。实验数据如表 1 所示。

表 1 仿真实验数据

任务数	虚拟机数	仿真次数	仿真算法
20	20	100	GA, CGA
50	20	100	GA, CGA
100	20	100	GA, CGA
200	20	100	GA, CGA

本文从任务完成时间和负载均衡度 δ 方面来验证该算法是否能提高服务器的利用率. 任务完成时间为最后一个任务完成时的时间减去第一个任务开始执行时的时间, 另外,

$$\delta = \sqrt{\frac{1}{m} \sum_{j=1}^m (Load_{j, \text{sum}} - Load_{\text{average}})^2} \quad (14)$$

式中, $Load_{j, \text{sum}}$ 为第 j 个虚拟机的总负载, $Load_{\text{average}}$ 为所有虚拟机的平均负载, 所以, 负载均衡度即为各虚拟机负载标准差. δ 越小则虚拟机负载波动越小, 负载越均衡.

为验证 CGA 算法的有效性, 将其与 Cloudsim 自带的遗传算法进行比较. 任务数分别取 20、50、100、200, 任务的长度取 2 000~10 000, 虚拟机的 MIPS 取 1 000~5 000. 此外, 令适应度函数中的 CPU 和内存的权重相等, 负载率与任务完成时间权重相等, 即 $w = v = 0.5$, $k_1 = k_2 = 0.5$, 交叉过程中元胞被选概率中的参数 $\lambda = 0.6$. 仿真 100 次, 取平均值作为最终结果来进行比较. 图 4、图 5 比较了随着任务数的增加, 两种算法执行所有任务所需的时间与负载均衡度.

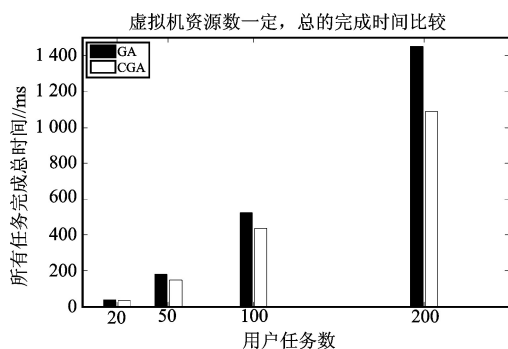


图 4 用户任务完成总时间对比

从图 4 和图 5 可以看出, 由于 CGA 算法结合了元胞自动机, 在遗传算法的遗传算子和交叉算子上进行了改进, 用户任务完成需要更少的时间, 且随着任务数的增加, 效果更明显, 负载均衡度值更小, 负载越均衡, 不会出现有些服务器过载, 而有些服务器闲散的问题, 所以, 该算法能提高服务器利用率.

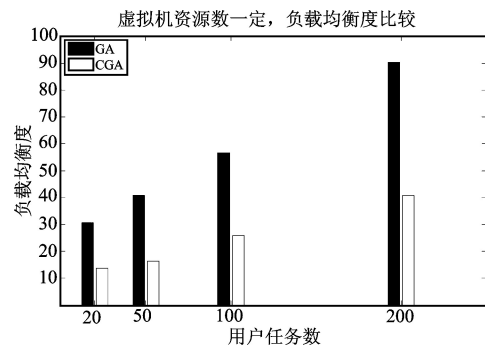


图 5 负载均衡度比较

6 结束语

提出符合云环境负载均衡的模型, 将其融入到遗传算法的选择和交叉算子上, 最终运用到云环境的负载均衡中, 在 Cloudsim3.0 云仿真平台上进行试验, 结果表明, 该算法可以缩短任务完成时间, 提高服务器的利用率. 但是, 算法本身还有许多地方可以改进, 下一步可以从变异算子进行改进.

参考文献:

- [1] 刘鹏. 云计算[M]. 北京: 电子工业出版社, 2011.
- [2] 张京钊, 江涛. 改进的自适应遗传算法[J]. 计算机工程与应用, 2010, 46(11): 53-56.
- [3] 王小平, 曹立明. 遗传算法—理论、应用与软件实现[M]. 西安: 西安交通大学出版社, 2002.
- [4] Kari J. Theory of cellular automata; a survey[J]. Theoretical Computer Science, 2005, 334(1/3): 3-33.
- [5] 赵超, 王晟. 基于云模型的负载均衡问题研究[J]. 微电子学与计算机, 2012, 29(3): 167-169.
- [6] Martin Randles, David Lamb. A comparative study into distributed load balancing algorithms for cloud computing[J]. IEEE, 2010(4): 551-556.
- [7] Alba E, Dorronsoro B. Cellular genetic algorithms[M]. Berlin: Springer-Verlag, 2008.
- [8] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, et al. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: Practice and Experience, 2011, 41(1): 23-50.

作者简介:

- 张水平 男, (1965-), 硕士, 教授. 研究方向为智能计算.
 庞敏 男, (1989-), 硕士. 研究方向为智能计算.
 仲伟彪 男, (1990-), 硕士. 研究方向为智能计算.
 何迎盛 男, (1990-), 硕士. 研究方向为智能计算.