面向多目标的云计算资源调度算法①

廖大强

(南华工商学院, 广州 510507)

摘 要: 在传统的虚拟机资源调度中, 仅仅考虑当前负载, 对虚拟机历史数据没有充分考虑, 在处理云计算资源调度的时候出现负载失衡的状况, 为了解决上述问题, 本文提出了基于启发式遗传算法的资源调度算法, 满足多目标规划的情况下实现云计算资源的调度. 算法在为用户提供服务的同时充分考虑虚拟机的各种开销和因素, 使提供云计算资源的服务器达到负载均衡. 对目前的负载情况和历史数据进行分析, 经过搜索和计算, 计算得到同时满足负载变化数据约束和最小动态迁移开销的最好的云计算资源调度方案. 最后, 通过仿真实验, 对算法进行验证, 通过引入负载变化率和平均负载距离二个性能参数来比较和衡量虚拟机负载. 实验数据证明, 所提出的算法具有很好的全局收敛性和资源利用率, 有效解决在资源调度中出现负载失衡和较大动态迁移开销的问题, 因此, 算法是可行和有效的.

关键词: 云服务; 资源调度; 遗传算法; 负载均衡; 迁移开销

Multi Objective Planning Research of Resource Scheduling Algorithm for Cloud Computing

LIAO Da-Qiang

(Nanhua College of Industry and Commerce, Guangzhou 510507, China)

Abstract: In the traditional virtual machine scheduling, we only focus on the current load, without fully considering the historical data in the virtual machine. As a result, we will suffer from load imbalance when scheduling the cloud computing resource. In order to solve that problem, this paper puts forward the algorithm of resource scheduling based on heuristic genetic algorithm, which can schedule the cloud computing resource while meeting the multi-objective planning. This algorithm fully considers various overheads and factors of virtual machine while providing service to users, so as to make the server, which provides cloud computing resource, achieve load balancing. By analyzing, researching and calculating current load and historical data, the writer induces the best scheduling scheme of cloud computer resource, which can meet the data constrains for the load variation and minimum dynamic migration overhead. Finally, by verifying the algorithm in a simulation experiment, the writer compares and measures the load of virtual machine by bringing in load change rate and two performance parameters of the average load distance. The experimental data shows that the proposed algorithm has very good global convergence and utilization rate of resources. It can solve the load imbalance and the large overheads of dynamic migration in the process of resource scheduling. Therefore, the algorithm is feasible and effective.

Key words: cloud services; resource scheduling; genetic algorithm; load balancing; migration overhead

1 引言

在云计算的虚拟机资源调度中, 包括了二种类型的调度模式, 分别为静态调度算法(离线调度)和动态调度算法(在线调度)[1]. 在第一种的云计算虚拟机资源调度中, 主要强调的是预先的调度方案, 在进行真正的调度之前就已经将相关的调度方案制定出来, 因此, 在调度之前就需要进行有效的分配, 在调度之后无法进行调整, 需要具有较强的与判断能力; 而对

① 基金项目:南华工商学院科研课题阶段性成果(15K03)
收稿时间:2015-05-30;收到修改稿时间:2015-07-17

© 中国科学院软件研究所 http://www.c-s-a.org.cn

于动态的云计算虚拟机调度算法中，可以在执行或在运行的过程中及时进行调整，因此，具有很好的执行效率和应用价值[2]. 在云计算的环境下，对于用户来说，物理节点是负载以及处理能力是很难进行估计和计算的，因此，需要在运行的过程中，及时收集相关的信息，充分的考虑到云计算的需求，及时的动态的调整云计算各个虚拟机的状态，根据历史的负载的数据和目前的负载状况来计算和调整云计算各个资源的调度，实现云计算虚拟机资源的动态的调度[3]. 如在某一个的时间段 T 中，对于虚拟机的请求是无法进行预知和计算的，无法知道预知究竟有多少的虚拟机请求的到达，因此，需要设计动态的云计算虚拟机资源调度，来对云计算资源进行动态的分配[4]. 下面是云计算虚拟机资源调度常见的几种策略，具体如图 1 所示[5].

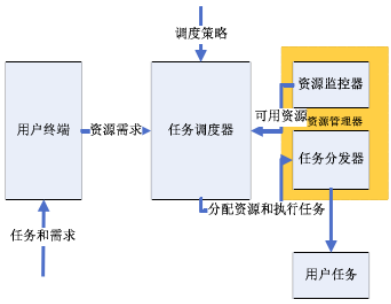

图 1 云计算资源调度概览

通过云计算方式可以很好的实现对服务器等各种资源的构建和整合，使得用户可以非常方便使用海量计算和存储的资源，对于上述的各种资源的整合和调度可以科学有效，并且可以不断适应用户的各种需求和变化，提高用户的良好体验，在这个过程中，需要合理分配各种资源和提供 Qos 保障. 在本文中，对云计算环境下的资源调度问题进行研究，通过对云计算资源调度的分析，对遗传算法进行改进并将其应用云计算的资源调度中，通过实验验证经过改进之后的算法可以达到较好效果.

2 资源调度建模

对于所有的物理服务器来说，需要建立衡量服务器负载的标准，可以通过物理服务器负载来表示，表示的物理服务器上的目前在运行各个虚拟机的全体负载的总和，在对物理服务器负载的衡量指标中，其中最重要的是处理器的负载指标[6].

在本文的研究中，对于物理服务器中的处理器的负载可以通过处理器负载因子的方式来对其进行权重的赋值，那么，综合负载 Load 可表示为:

$$Load=(W_{cpu},W_{mem},W_{disk},W_{bandwith})\begin{pmatrix}W_{cpu},\\W_{mem}\\W_{disk}\\W_{bandwith}\end{pmatrix} \quad (1)$$

在上述的公式中某个的处理器的负载，可以表示为 Wcpu，其他因素可以表示为 Wmem，Wdisk，Wbandwidth. 在实际的云服务环境下，包括很多其他的衡量指标，如内存、硬盘、带宽，在其中处于最重要的是处理器的性能指标，为了研究方便，在本文中，选取处理器作为物理服务器的衡量物理服务器负载的指标.

2.1 物理服务器历史负载表示

前面对于服务器的当前的负载情况建立了数学表示模型，对于服务器的历史的负载信息也可以通过数学描述来表示，设定在某一个的时间的监控时间节点为T，那么表示的是在目前的时间节点作为参照，往前推 T 个时间段所得到的这段时间内的历史负载的信息. 与此同时，也可以根据实际情况，将上述的 T 时间段内的一整段时间进行划分，经过划分得到 n 时间段，因此，对于历史的时间段 T 可以表示为 T=[(t1-t0)，…，(tn-tn-1)]. 其中(tk-tk-1)表示第 k 个时间段.

下面对服务器的历史负载信息进行数学模型的建立，在这一段的时间内的平均负载可以表示为:

$$\overline{LV(i,j,T)}=\frac{1}{T}\sum_{k=1}^{n}LV(i,j,k)*(t_k-t_{k-1}) \quad (2)$$

在上述公式中对虚拟机 V_{ij} 的处理器的使用情况计算记物理节点 P_i 上的第 j 个虚拟机 $V_{ij}(1, j, m_i)$在第 k 个时间段的负载为 $LV(i,j,T)$ = v_{CPU} * $U(t_k)$，在 t_k 时间段内的处理器的使用率可以表示为 $U(t_k)$.

对于物理服务器来说， Pi 在时间跨度 T 上的负载总和为:

$$LP(i,T)=\sum_{j=1}^{m}\overline{LV(i,j,jT)} \quad (3)$$

设定目前的虚拟机的分配为 V，经过调度之后对于物理机来说其负载可以通过下面的公式进行计算，具体如式(4)所示.

$$LP(i,T)l=\begin{cases}LP(j,T)+LV(V) & \text{当i = j时}\\ LP(i,T) & \text{其他}\end{cases} \quad (4)$$

在上述的公式中，虚拟机负载表示为 LV(V)，对

© 中国科学院软件研究所 http://www.c-s-a.org.cn

于虚拟机 v 经过分配之后被调度到物理机为 Pj, .

在虚拟机被分配之后, 设定虚拟机 v 经过调度被分配到物理机为 Pj, 对于这个物理机来说, 在后面的运行中可能会出现变化, 需要进行重新的调度, 实现资源调度, 其映射的方案为 S_j 在 T 时段的负载变化用标准差可以通过如下式(5)所示:

$$\sigma(s_j=T)=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\overline{LP(T)^{''}}-LP(i,T)^{''}_{i})^2} \tag{5}$$

上述的公式就很好的表示的当前的物理服务器的负载以及出现改变之后的服务器的负载估算.

2.2 物理服务器负载变化量

首先是物理服务器的负载的变化量的计算, 在这个性能指标的计算中, 这台物理服务器的负载的变化量可以通过如下的公式进行计算, 具体计算的公式如式(6)所示.

$$\sigma(s_j=T)=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\overline{LP(T)^{''}}-LP(i,T)^{''}_{i})^2} \tag{6}$$

在上述公式中, 在时间 T 内的对虚拟机 V 进行的动态分配的方案为 Sj, 那么在式(6)中,

$$\overline{LP(T)^{''}}=\frac{1}{N}\sum_{i=1}^{N}\overline{LP(T)^{''}}$$

通过上述的计算就可以计算得到物理服务器的负载变化量的大小.

经过上述的计算可以得到物理机器的负载变化量大小, 并且根据历史负载的数据来制定资源调度的映射方案, 通过数学模型来表示, 同时也充分考虑到在动态迁移过程中出现的迁移代价, 通过代价因子 $\rho(S^*)$ 来计算其大小, 为动态迁移提供参数.

在上述所得到各个约束条件中, 得到云服务调度映射的方案, 可以通过如下的集合来表示$\{S^*\}=\{S_1^*, S_2^*, \ldots, S_D^*\}$, 那么整个调度的设计目标就是在上述的集合中计算和比较得到最优的映射方案, 通过这个方案来实现云服务服务器的资源调度, 这个过程实际上就是计算和搜索得到最小化负载平衡迁移代价因子.

2.3 负载均衡映射方案

设定云服务中的服务器的数量为 N 个, 那么就可以构建服务器的集合 $P=\{P_1, P_2, \ldots, P_N\}$, 对于每一个物理服务器上的虚拟机也可以通过构建得到一个集合 $\{V_1, V_2, V_3 \ldots V_m\}$, 其中 V_i 是指第 i 台物理机器的系统中的所有物理服务器集合为 $P=\{P_1, P_2, \ldots, P_N\}$, 其中 N 表示物理服务器数量, $P_i(1, i, N)$表示第 i 台物理服务器. 我们将物理服务器 Pi 上的虚拟机集合设为$\{V_{1i}, V_{2i}, V_{3i}, V_{mi}\}$, 这里 mi 表示物理服务器 P_i 上的虚拟机数量. 用 S_{ij} 表示将虚拟机请求队列中的第 i 台虚拟机映射到第 j 台物理机 P_j 上的映射方案. 在动态迁移技术中在某一个时间 T 内, 对于新出现的请求往往可能不仅仅只有一个, 因此, 可以通过建立一个请求的队列来表示, 具体这个队列的表示为 S. 在这个时间段内, 已经分配了 k_0 个云服务资源调度, 当出现新的请求后, 这个请求为 k_1 个, 那么总的分配请求为 k_0+k_1 个, 组成整个资源调度的映射方案.

2.4 迁移代价估计

在动态迁移中需要一定的系统资源的开销, 如果这个开销不小的话, 那么也是重要的性能指标, 对于迁移代价的估计计算成为重要的内容.

对于迁移代价估计也可以通过数学模型来表示, 在分配方案 S*中, 分别定义虚拟机数量 M'与虚拟机总数 M, 那么其的比值是代价因子 $\rho(S^*)$ 可以通过如下的公式可以表示, 具体如式(7)所示.

$$\rho(S^*)=\frac{M^{''}}{M} \tag{7}$$

3 算法设计及流程

在以往的遗传算法中, 对于算法的运行和使用的效率起到非常重要应先处理的是种群的选择, 如果对参数选择不是很好的话将会直接影响到最后的算法效果. 例如选择算子当中的选择机制产生出不同问题和缺陷, 利用一定的比例的方式来进行选择操作, 这样就比较容易的导致在选择的种群当中出现超强个体的时候, 对于上述的超强个体将可能会影响到其他区域的种群, 甚至可能会充满到其他的种群中, 这样就可能会降低种群的多样性. 下面对遗传算法的资源调度进行详细的算法设计.

3.1 初始化种群

在基于遗传算法的虚拟机资源调度的树形结构中, 对于树形中每一个对象都有其具体的节点, 在节点中可以包括各种的函数或相关的指令, 并且形成具有先后序列的基因编码, 通过遗传算法来解决问题时, 首先要对问题进行编码. 编码方式应当根据问题的特点和遗传算子的设计方式来选择. 遗传算法用二进制来编码基因染色体结构. 通过观察发现, 本文中的物理服务器和虚拟机是一对多的关系, 因此我们选用树结

© 中国科学院软件研究所 http://www.c-s-a.org.cn

构来表示基因染色体. 也就是说, 每个映射方案用一棵树来表示. 第一层是系统调度和管理节点作为根节点, 第二层所有的 N 个节点表示物理服务器, 第三层的 M 个节点表示运行在该物理服务器上的虚拟机. 生成树初始化定义具体如下:

1) 在生成树元素中主要是有物理机和虚拟机二个元素来组成, 由上述的二个元素来组成;

2) 在生成树中, 位于最上层的根节点是云服务资源调度的调度节点;

3) 在云服务中的全部的物理机和虚拟机都在这个生成树中;

4) 在生成树中的位于叶子位置的全部是虚拟机元素.

初始化种群的流程可以表示如图 2 示.

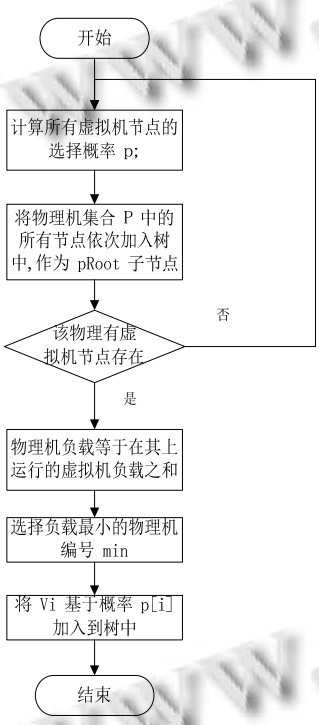

图 2 初始化种群的流程图

对于初始化种群来说, 在经过生成树和进化之后就可以获取得到物理机与虚拟机之间的映射关系. 在进化的过程中, 其进化的选择概率 p 可以通过如下的方式来进行计算, 通过计算每一个虚拟机的负载与全部虚拟机的负载的比值, 这样就可以计算得到选择的概率 p; 接着, 对于每一个的虚拟机来说, 根据前面的选择的概率 p 来对虚拟机进行分配, 对于虚拟机来说按照概率 p 将其分配到最小负载的物理机器上, 经过这样的分配之后就可以得到生成树的叶子节点, 与此类推, 通过前面的方式就可以很好的将虚拟机全部的分配出去, 这样有个好处就是, 将部分被选中机会较小的虚拟机都有可能被选中.

3.2 适应度计算

在前面经过对云服务资源调度进行编码, 形成生成树结构的物理机与虚拟机的映射关系, 并且在生成树的过程中利用初始化种群实现物理机与虚拟机关联, 接下来需要计算遗传算法中的适应度函数, 通过适应度函数来对种群进行进化的约束, 并且推动种群往好的方向进化. 负载约束条件进行数学描述, 在这个负载约束中定义了其数学模型可以通过符号 σ 来表示, 通过这个数学模型可以对资源调度的情况进行描述.

在云服务资源调度中, 对于 σ 的数值, 如果其的数值越小的话, 那么资源调度的效果越好. 在进行适应度函数计算的时候要求适应度的函数越大越好, 跟前面的负载约束定义是相反的, 因此, 对于这二个数学描述需要一定的修正.

对于在云服务的资源调度中, 对于种群的适应度函数的计算可以通过如下的方式来进行, 具体如式(8)和式(9)所示:

$$f(S,T)=\frac{1}{A+B*f_H} \tag{8}$$

$$f_H=\Phi(\sigma(S,T)-\sigma_0),\Phi(X)=\begin{cases}1, X\le 0\\ c, X>0\end{cases} \tag{9}$$

在上述的对与计算资源调度的适应度计算公式可以看到, 对于适应度函数中的符号 A,B 是应用的权重系数, 由其重要程度来定义; 对于(4-2)中进行适应度函数计算的公式中, 其中 σ (S,T)在前面已经有定义, 表示的是在 T 时间内的负载变化标准差; 云服务资源调度的负载变化约束条件是由 σ_0 来进行定义, 具体在本文的研究当中, 负载变化约束的范围定义为 $\sigma_0=0.5$; 还有就是惩罚函数, 在本文中通过如下的方式进行定义, 如果符合要求的话, 那么惩罚函数的数值将设置为 1, 否则的话, 惩罚函数的数值设置为 c. 对于惩罚函数的数值的设置, 在不同的应用环境下的数值有所不同, 具体在本文中的惩罚函数数值设置 c=1.5.

3.3 选择操作

对于本文对于云服务资源调度的研究中采取的选择策略是基于适应度的选择策略, 在这个策略中根据

© 中国科学院软件研究所 http://www.c-s-a.org.cn

适应度比例选择算法来选择个体, 对于上述的算法, 下面进行分析和介绍.

step1: 对于目前的个体计算其的适应度函数的数值;

step2: 对上述的经过计算所得到的个体中选取适应度数值最大的个体, 并将其复制到下一代的个体中, 在这个基础上计算个体的选择概率, 具体计算的方式如式(10)所示:

$$P(S_i)=\frac{f(S_i,T)}{\sum_{i=1}^{D}f(S_i,T)} \tag{10}$$

在上述计算个体选择概率的公式中, 在种群中的第 i 个个体的适应度由 $f(S_i,T)$ 来表示, 种群的规模大小由 D 进行定义, 通过上述的公式就可以计算得到个体的选择概率.

step3: 根据上述的公式采取轮盘赌选择进行个体的选择, 通过轮盘赌选择就可以得到具有比较高适应度函数数值的个体, 当然通过轮盘赌选择可可能选择得到适应度个体较低的个体.

通过轮盘赌选择的策略中, 可以看到对于轮盘来说, 划分为若干个区域, 表示为D, 实际上是种群规模大小; 每一个区域都是一个扇区, 对于某一个的个体 i 来说, 其的范围大小可以通过圆心弧的面积计算得到; 在选择的时候, 产生随机数表示为 k, 并且满足约束 $p1+p2+...+pi-1<k \quad p1+p2+...+pi$, 在这个计算方式中, 对于个体来说, 如果其的适应度越大的话, 那么其的扇形的面积就相对大些, 从选择的概率来说, 就有可能被轮盘进行选中.

上述的轮盘赌的选择策略可以通过如下的程序来实现:

Step1: 通过策略得到随机数 r, 并且这个随机数的产生是均匀和满足[0, 1]条件的;

Step2: 如果满足条件的话 $q_{k-1}<r<q_k(2\le k\le n)$, 那么对于种群中的染色体 x_k 将会被选中. 在上述的条件中, 对于染色体的积累概率可以通过式(11)来进行计算和定义:

$$q_i=\sum_{i=1}^{i}P(x_{)i}) \tag{11}$$

3.4 交叉变异

在树形编码的交叉操作中, 首先在双亲中选择交叉点, 通过交叉点就可以获取得到结成亲代的子节点, 树形编码交叉中对 ParentA 来说, 通过断裂右叉树得到子节点, 对 ParentB 来说, 通过断裂左叉树, 得到另外的子节点, 接下来就是二个子节点进行交换产生新的子代.

在资源调度的交叉操作中, 根据前面的树形交叉得到如下详细的交叉过程:

Step1: 通过前面的选择策略, 利用轮盘赌的方式得到具有适应度较高的亲代个体, 分别表示为 T0 和 T1.

Step2: 对于上述的二个个体来说, 经过组合形成个体 T2, 经过组合之后就可以得到如图 4-8(a, b)中所示的树形结构, 并且在这个结构中保留了相关的叶子节点; 对于叶子节点计算其的负载根据负载的情况得到选择概率p, 通过p将叶子节点分配到物理机负载最小节点, 经过上述的方式就可以将叶子节点全部分配.

Steps3: 不断对前面的个体重复上述的过程, 一直到 D*Pc 的个体分配完.

在遗传算法中, 不仅仅需要选择较高适应度的个体染色体, 还需要部分的染色体的变异, 这样就可以实现一方面种群多样性, 另一方面可以防止部分的个体早熟情况出现, 使得进化往更加优越的方向发展. 在遗传算法中, 如果采取变异的操作可以使得算法能够在局部进行搜索操作得到最优解, 具体到本文所研究的云服务资源调度中, 采取变异的操作通过如下的公式来计算其变异的概率, 具体如式(12)所示.

$$P_m=\exp(\frac{-1.5\times 0.5t}{D}*\sqrt{M} \tag{12}$$

在上述的计算云服务资源调度的变异概率中, 主要是通过自适应变异方式来实现, 其中在公式中的种群规模由符号 D 来表示, 虚拟机的规模通过符号 M 表示, 在种群中的个体在将实现变异的时候通过随机选择方式实现, 在本文对于基因重复变异的防止变异, 在进行变异的时候, 如果非叶子出现变异将叶子节点也随之进行改变.

在进行变异的时候, 通过前面计算得到的变异概率选取二个个体作为变异的对象, 随着选取物理机得到物理机为p1和p2; 接着对上述的物理机上面的虚拟机也就是树形编码中的叶子节点的数量为为M1和M2, 在上述的基础上产生随机数为 V1 和 V2, 对于这个随机数来说需要满足[1, M1]和[1, M2], 所产生的随机数就作为上述树形的编号, 最后对上述的叶子节点进行

© 中国科学院软件研究所 http://www.c-s-a.org.cn

交换.

3.5 终止条件

在本文的研究中, 对于基于遗传算法的云服务资源调度的终止条件为二个, 分别为个体符合负载均衡约束条件, 还有就是遗传的迭代次数满足定义的代数, 只要满足上述的某个条件就可以退出遗传算法的运行. 接着, 对上述的二个退出遗传短发的方案进行比较, 分别计算其迁移开销最小的方案, 选取作为云服务资源调度的的映射方案.

3.6 算法流程

通过上述的算法要求和步骤就可以实现全局的云服务资源调度算法, 算法的详细过程如图 3 示.

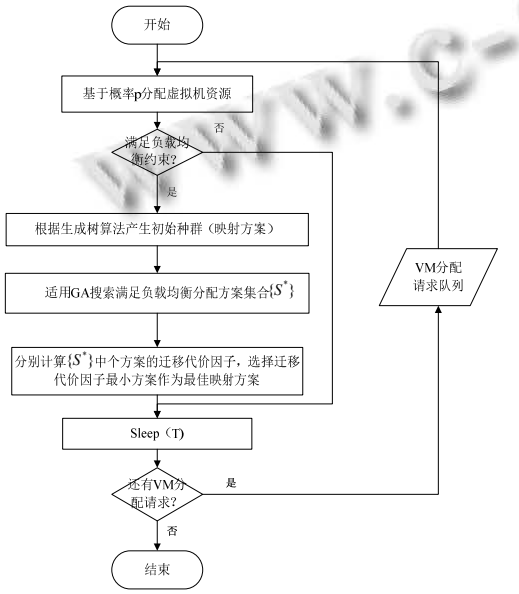

图 3 全局调度策略流程图

通过前面的实现步骤的总结, 具体其实现的步骤如下:

Step1: 在进行遗传算法的计算初始化的时候, 在系统中还没有任何的虚拟机资源可供调度, 因此也就没有以往物理机的负载数据信息; 当存在云服务资源的调度时候, 根据某个虚拟机与全部虚拟机负载的比值来进行分配资源, 对其中属于负载最小的物理机进行资源调度;

Step2: 当运行一段时间后虚拟机的数量将逐渐增加, 因此, 需要进行分配资源的虚拟机也不断上升, 在这时候计算云服务中的所有物理机在时间 T 范围内的平均负载和负载变化情况, 如果不符合负载约束条件的话, 将通过生成树算法得到初始化种群, 计算得到{S0}, 在上述基础上计算得到符合要求的方案, 可以表示为{St};

Step3: 对于上述通过生成树算法计算得到映射方案进行计算迁移代价因子, 接着选取属于代价因子最小的作为最后的云服务资源调度方案;

Step4: 以此类推, 以后出现新的云服务资源调度也按照上述的方案, 直接转到 Step2 计算符合要求的映射方案.

在算法的实现中, 采取的是 C++面向对象语言进行描述和实现, 在实际的运行过程中, 通过遗传算法的云服务资源调度可以很好实现最佳的云服务虚拟机资源的调度, 同时, 经过不断的运行, 可以得到各个映射方案的累计, 形成累计效应, 这样就可以在下一次的云服务资源调度的时候较快的寻找到资源调度的映射方案, 甚至在出现意外的需要大负载的请求, 调度算法可以在较短的时间范围内可以计算得到最小迁移代价的负载均衡方案, 从而逐渐实现云服务的资源调度.

4 仿真结果

4.1 仿真环境设定

在本文的利用遗传算法进行云服务资源调度中, 采取的实验平台是开源的云服务框架平台 Hadoop. 在本文的利用遗传算法进行云服务资源调度中, 采取的实验平台是开源的云服务框架平台 Hadoop. 通过这个开源云服务测试平台可以很好的实现云服务资源的调度和实验数据的收集分析. 对于 Hadoop 来说, 其运行需要一定的硬件环境, 在本文中选取一台高性能的服务器搭建 Hadoop, 选取 6 台安装客户端的物理机作为测试的节点, 上述的测试服务器都是在局域网内实现的, 在虚拟机的生成中, 通过 Ubuntu-Linux 技术就可以快速的生成各个虚拟机的映像, 通过上述的方式就可以搭建得到前面树形结构图. 硬件的详细性能配置如下表 1 所示.

表 1 实验服务器的硬件配置表

服务器	硬件配置
调度管理节点	Intel Core i5™ 2.5GHz CPU, 4GB 1600MHz DDR3 RAM
虚拟机客户端	Intel Core i5™ 2.5GHz CPU, 4GB 内存, 100GB 硬盘
网络	局域网
切换	KVM

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4.2 仿真实验设计

接下来对仿真实验进行设计, 对算法收敛性、算法效率进行实验设计.

(1)算法收敛性实验设计

下面对遗传算法的云服务资源调度进行仿真实验, 计算在云服务资源调度环境下遗传算法的收敛性能. 在本文的研究中, 设定有 5 台的物理机, 在这些的物理机中运行 16 台虚拟机, 具体虚拟机的部署和机构如图 4 所示.

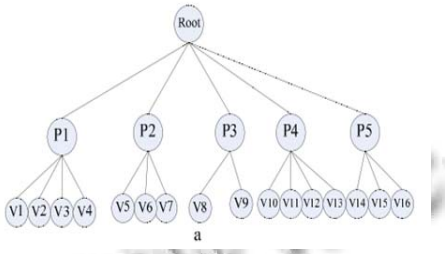

图 4　使用算法前的映射关系

接着是在 T 时间内的虚拟机的平均负载, 对于在 T=1 的时间内的虚拟机的平均负载具体如图 5 所示.

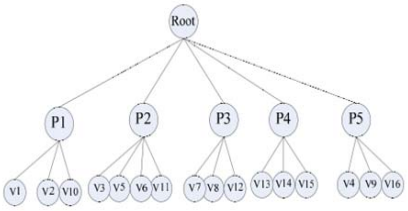

图 5　使用本文算法后的映射关系

在这个示意图中可以看到, 对于物理机 p1 来说, 运行有 V1、V2、V3 和 V4 的虚拟机, 对于 P2 的物理机运行有 V5、V6、V7, 对于物理机 P3 运行 V8、V9 的虚拟机, 对于 P4 的物理机运行的虚拟机有 V10、V11、V12、V13, 对于物理机 P5 来说, 运行的虚拟机有 V14、V15 和 V16,经过实验数据的采集可以获取得到如下的虚拟机的计算资源的负载情况, 具体来说如图 6 所示.

在上述的数据中可以看到, 对于每一个的物理机来说, 其的负载是不尽相同, 对于在上面运行的虚拟机来说, 其的负载水平也是不尽相同, 因此, 需要进行云服务资源的调度实现负载均衡.

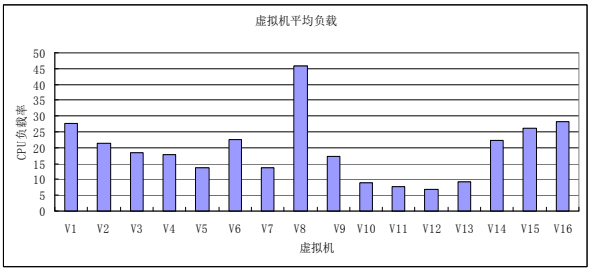

图 6　虚拟机平均负载

为了研究方便, 在本文的研究中, 设定种群 D 的数值为 50, 在遗传算法中的选择概率的 Pr=0.1, 交叉概率: Pc=0.9, 对于变异的概率来说, 其是可以自适应的, 并且分析在上述的各个参数的配置中, 交叉概率 Pc∈[0,1], 变异概率 Pm∈(0,1), 对于负载的变化数值来说, 具有一定的约束 0.50, 通过上述的计算就可以得到云服务环境下的资源调度的映射方案, 具体如图 5 所示.

在经过云服务资源调度之后的物理机与虚拟机的映射方案中可以看到, 通过调度可以将虚拟机的负载更加均衡, 对于每一个的物理机的节点来说, 其的负载水平趋于相同, 可以达到相当的程度, 实现了负载的均衡. 从物理机的负载的变化率来看, 其的变化符合要求.

从前面的通过虚拟机在实验前后的变化和映射关系可以看到, 通过遗传算法可以使得物理机与虚拟机的负载水平达到均衡, 算法具有很好的全局收敛性能, 算法可以再较短的时间内实现收敛达到最优解的目的.

(2)算法效率实验设计

从前面算法的收敛性能的分析可以看到, 算法具有较好的全局收敛性, 在很短的时间就可以计算得到最优解, 对于云服务资源调度算法来说, 不仅仅需要关注算法的收敛性能, 还需要分析算法的效率. 在云服务的环境下, 通过大量的物理机与云服务资源的调度计算, 计算最优的调度映射方案下处理器的执行时间就可以衡量算法的效率.

通过最优调度方案的 CPU 执行时间与虚拟机数量关系的示意图可以看到, 随着数量增加算法的性能仍能保持稳定, 证明算法具有良好的效率, 具体如图 7 所示.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

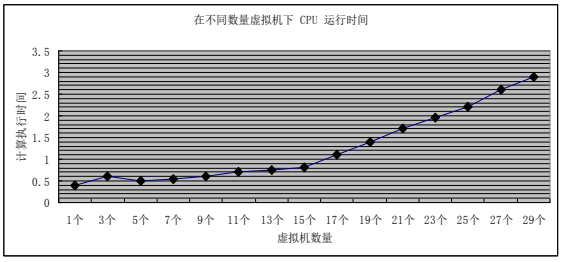

图 7 在不同数量虚拟机下 CPU 运行时间

在本文的研究中, 对于基于遗传算法的云服务资源调度算法中, 算法的效率的实验环境跟前面是类似的, 通过增加虚拟机的数量来计算处理器的执行时间, 在最优映射方案情况下的最优解的计算时间, 经过仿真实验, 随着虚拟机的数量的增加, 其最优调度方案的 CPU 执行时间也随着增加, 但是增加的幅度逐渐减少, 增加的趋势逐渐趋于平稳, 总体的执行时间并没有增加很多.

此外, 对稳定性能进行计算, 通过虚拟机的负载变化率来进行计算, 其可以通过数学符号 α 来进行描述, 其具体的计算公式如式(13)所示.

$$\alpha = \frac{|L(t) - L(T_0)|}{L(T_0)} \tag{13}$$

在上述的公式中, 其中的 $L(T_0)$是最开始的负载情况, $L(t)$则表示的是目前的负载水平.

为了验证基于遗传算法的云服务资源调度算法中的稳定性在本文采取与其他的调度算法进行比较, 选取了最小负载调度(Hadoop 中的尽力交付策略)和轮转调度算法(Hadoop 里的预先订制策略)这二个具有代表性的云服务资源调度算法进行算法的比较.

在实际的应用中, 在某一个瞬间内的服务器的负载并不能真正的反映服务器物理机的实际负载状况, 如服务器物理机在进入相对较多的情况下, 那么对于物理机的负载来说实际上是比较高的, 如果有大量的虚拟机退出, 也会造成服务服务器在某一个的瞬间使得服务器的负载较低, 实际上述的瞬时负载并不反映服务器的真实负载水平. 为此, 在本文中提出了平均历史负载率, 通过对每一段时间内的服务器的负载水平和变化状况, 通过对历史的负载水平数据进行分析, 结合当前的服务器的负载估算就可以得到服务器的真实的负载情况, 最后对调度可能产生的影响进行预计, 通过预算就可以很好的避免瞬时负载计算出现的问题, 在上述的基础上, 将虚拟机进行调度, 将其安排分配到堵在水平相对较小的服务器上, 同时需要计算迁移代价因子, 计算其大小, 这样就可以得到最优的调度映射方案, 保证系统的负载均衡, 使得后续的调度都可以保持系统的负载均衡.

4.3 结果与分析

(1)算法的负载稳定性.

在本次的实验中, 通过云服务开源平台生成 100 台的虚拟机器, 每一个的处理器的负载情况设定为 5%到 30%. 然后计算其的负载水平, 具体结果如图 8、9、10 所示.

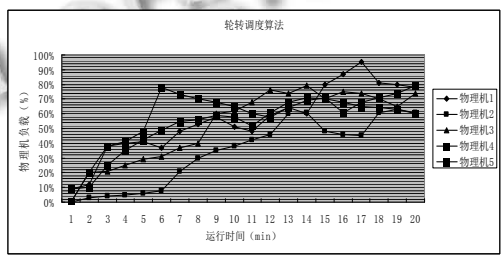

图 8 采用轮转调度算法系统负载变化

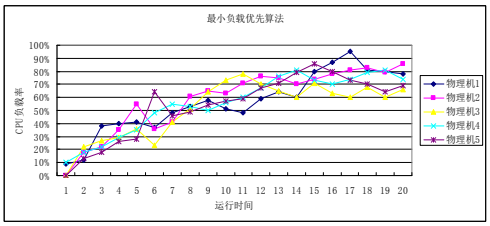

图 9 采用最小负载优先调度算法系统负载变化

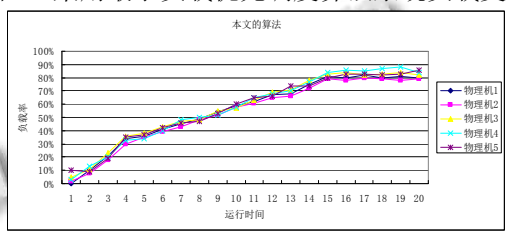

图 10 采用本文调度算法系统负载变化情况

在上述的三个不同算法的系统运行时间与物理机负载之间关系可以看到, 即便出现负载出现变化, 但是本文的算法能够使得物理机在较短时间内实现负载的均衡, 并且负载变化的幅度不会很大.

(2)负载变化率.

在本文的研究中, 引入负载距离 LD 的因素, 通过这个因子来对五个物理机的变化情况进行描述, 如果物理机之间的间隔越小的话, 那么其的负载距离 LD 越小, 说明算法对于物理机负载效果越好, 物理机就更加的均衡. 接下来, 设定不同的观察点, 对物理

© 中国科学院软件研究所 http://www.c-s-a.org.cn

机在不同的点之间的负载进行观察, 设定观察点的负载为 L_0, L_1, L_2, L_3, L_4, 通过上述的观察点作为参照, 计算负载距离 LD 的大小, 其计算的公式为:

$$LD_p = \sum_{i=0}^{3} \sum_{j=i+1}^{4} \left| L_i - L_j \right| \tag{14}$$

经过计算之后, 就可以得到不同点的平均负载距离, 具体结果如图 11 所示.

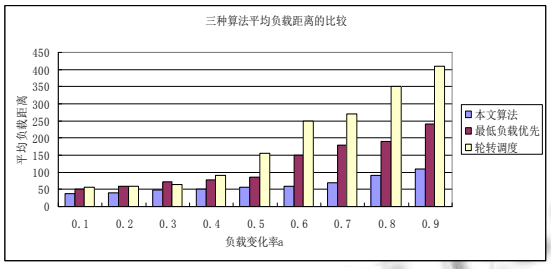

图 11　三种算法平均负载距离的比较

(3)迁移代价.

在本文提出的算法中, 对于动态迁移实现调度后的迁移开销较小. 最小负载优先调度算法在调度实现负载均衡过程中可能会出现较大的迁移代价, 甚至也有可能在动态的迁移之后出现负载的部均衡, 造成相反的效果. 图 12 反映了当虚拟机负载变化率变化时虚拟机的迁移比例. 从图 12 可以看出, 本文使用的算法极大地减小了迁移代价.

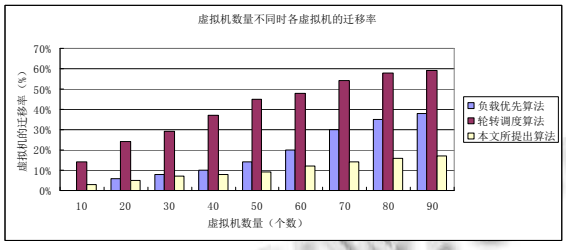

图 12　虚拟机数量不同时各虚拟机的迁移率

(4)资源利用率.

云服务资源调度实现系统的负载均衡, 在这个过程中可能需要消耗一定的系统资源, 为此, 通过引入资源利用率来衡量算法的优劣. 经过实验, 得到如下的实验数据, 具体如图 13 所示.

在上述的不同算法之间的资源利用率的示意图可以看到, 在本文的算法中可以很好的实现对资源的利用, 在不同数量的虚拟机的环境下都可以保持较大的利用水平, 并且保持较高的资源利用率达到 75%以上的水平. 因此, 算法具有高资源利用率, 可以在一定程度上节省资源.

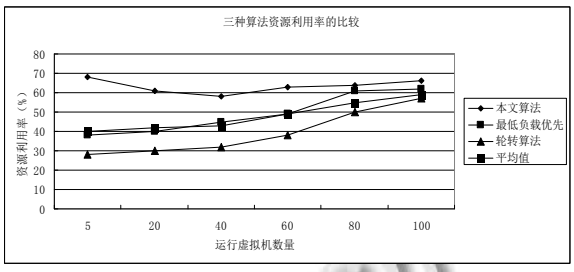

图 13　三种算法资源利用率的比较

通过前面的分析, 可以看到算法具有很好的全局收敛性能, 在较短的时间内实现收敛达到最优解的目的. 接着进行负载变化率对负载均衡的影响、迁移代价影响、算法的整体利用率进行分析, 通过仿真实验, 验证算法是可行和有效的.

5　结语

云服务资源的调度算法中通过引入树形编码, 具有较好的适应度策略和适应变异率, 使算法具有良好的收敛性能, 通过仿真实验, 验证算法是具有较高的资源利用率和较低的迁移代价, 可以应用到云服务的虚拟机资源调度中.

参考文献

1 Fox A, Griffith R, Joseph A, et al. Above the clouds: a Berkeley view of cloud computing[Thesis]. Berkeley: University of California, 2013: 28–34.

2 Buyya R, Yeo CS, Venugopal S, et al. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 2011, 25(6): 599–616.

3 廖大强,邹杜,印鉴.一种基于优先级的网格调度算法.计算机工程,2014,40(10):11–16.

4 韩建民,鹿玲杰.资源调度机制在计算机控制系统中的应用.计算机应用,2013,41(2):5–6.

5 何福贵,侯义斌.基于有限优先级的动态调度分组算法.北京工业大学学报,2013,34(8):11–12.

6 涂刚,阳富民.基于动态优先级策略的最优软非周期任务调度算法.计算机研究与发展,2014,42(11): 23–24..

7 Hovestadt M, Kao O, Kliem A, et al. Adaptive online

© 中国科学院软件研究所 http://www.c-s-a.org.cn

compression in clouds-making informed decisions in virtual machine environments. Journal of Grid Computing, 2013, 47(1): 41–57.

8 强彦,卢军佐,裴博.基于决策树分类的云资源调度算法研究与实现.太原理工大学学报,2012,43(6): 715–718.

9 邓景文.集群系统下面向用户的作业公平调度算法[硕士学位论文].北京:北京邮电大学,2013.

10 柳少锋,董剑.一种基于优先级队列的集群动态反馈调度算法.智能计算机与应用,2012,12(4):45–49.

11 廖大强,邬依林,印鉴.基于禁忌搜索算法的线路规划方案求解.计算机工程与设计,2015,36(5):1368–1472.

12 李小六,张曦煌.虚拟化云计算数据中心能量感知资源分配机制.计算机应用,2013,39(12):79–86.

www.c-s-a.org.cn

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn