软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(Suppl.(2)):201−212

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Xen 虚拟机资源调度优化算法*

乔百友[1,2], 蔡仁翰[2], 陈东海[1], 王 虹[1], 陈 洋[1], 王国仁[1,2]

[1](东北大学 信息科学与工程学院,辽宁 沈阳 110819)

[2](国家海洋信息中心,天津 300171)

通讯作者: 乔百友, E-mail: qiaobaiyou@ise.neu.edu.cn, http://www.neu.edu.cn

摘 要: 针对 Xen 虚拟化平台中虚拟机资源分配不合理的问题,提出了两种资源调度优化算法,即细粒度优化算法和粗粒度优化算法.细粒度优化算法主要解决单个物理节点上虚拟机资源分配不合理问题,能够根据物理节点上运行的各虚拟机的资源利用情况来调整资源分配量,适当增加利用率较高的虚拟机的资源,减少资源利用率低的虚拟机的资源,从而优化资源分配,提高资源利用效率,避免不必要的虚拟机迁移.粗粒度优化算法是针对集群中多个物理节点之间虚拟机负载不均衡问题而提出的.该算法结合粒子群优化技术,选择将集群系统中热点物理机上的部分虚拟机迁移到最适合的冷点物理机上,从而避免高载物理机宕机.实验结果表明,这两种资源调度优化算法能够有效解决虚拟机资源分配不合理的问题,具有较好的适用性和应用前景.

关键词: Xen;虚拟化;粒子群;资源调度优化算法

中文引用格式: 乔百友,蔡仁翰,陈东海,王虹,陈洋,王国仁.Xen 虚拟机资源调度优化算法.软件学报,2014,25(Suppl.(2)):201−212. http://www.jos.org.cn/1000-9825/14038.htm

英文引用格式: Qiao BY, Cai RH, Chen DH, Wang H, Chen Y, Wang GR. Resource scheduling optimization algorithm for Xen virtual machines. Ruan Jian Xue Bao/Journal of Software, 2014,25(Suppl.(2)):201−212 (in Chinese). http://www.jos.org.cn/1000-9825/14038.htm

Resource Scheduling Optimization Algorithm for Xen Virtual Machines

QIAO Bai-You[1,2], CAI Ren-Han[2], CHEN Dong-Hai[1], WANG Hong[1], CHEN Yang[1], WANG Guo-Ren[1,2]

[1](College of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

[2](National Marine Data and Information Service, Tianjin 300171, China)

Corresponding author: QIAO Bai-You, E-mail: qiaobaiyou@ise.neu.edu.cn, http://www.neu.edu.cn

Abstract: Arming at the irrational resource allocation problem in the Xen virtualization platform, this paper proposes two resource scheduling optimization algorithms: the fine-grained algorithm and the coarse-grained algorithm. The fine grained algorithm is mainly for resource allocation of single physical node, which dynamically adjusts the allocated resource amount of each virtual machine according to its resource utilization, and appropriately increases the resource amount for the virtual machines whose resource utilization are high and reduces the resource amount for those whose resource utilization is low, thus improves resource utilization efficiency and avoids unnecessary virtual machine migrations. The coarse-grained algorithm focuses on the load imbalance problem among multiple physical nodes in a cluster, and applies the particle swarm optimization technique to select some virtual machines on the hot physical machines and then immigrates it to the most suitable cold physical machines in a cluster system, thereby solving load imbalance problem of the cluster system and avoiding high load physical machines downtime. Experiments show that the proposed two scheduling optimization algorithms can effectively solve the resource allocation irrational problem of virtual machines and have better adaptability and application prospects.

Key words: Xen; virtualization; particle swarm optimization; resource scheduling optimization algorithm

* 基金项目: 国家自然科学基金(61073063, 61173029, 61173030, 61100028, 61272182); 国家海洋公益性项目(201105033); 数字海洋国家重点实验室开放基金(KLDO201306)

收稿时间: 2014-05-07; 定稿时间: 2014-08-19

虚拟化技术是伴随着计算机的产生而出现的,并随着云计算技术的发展而得到了越来越广泛的应用.服务器虚拟化是虚拟化的核心技术,它通过让一个物理节点同时运行多个相互隔离的虚拟机,从而满足不同需求,提高资源使用效率.然而,由于虚拟机对硬件资源的需求各不相同,并随着运行任务的动态变化而变化,其资源占用会出现严重的不均衡,有的虚拟机占有过多硬件资源但却利用率十分低,而有的虚拟机资源利用率相当高,但其硬件资源占有量却极少,这必然严重影响整个系统的运行效率.在一个动态的云计算集群系统中,虚拟机的数量和运行情况都不固定,并随着用户的申请和释放而不断变化,集群中各个物理节点上的负载或高或低,甚至会出现严重的负载不均衡情况,有可能造成高负载机器宕机,严重影响系统的可用性.因此,如何合理地进行资源调度,保证虚拟机资源合理分配及整个集群系统的负载均衡,是一个十分重要的问题.

针对虚拟机资源调度问题,研究者提出了多种资源调度解决方案和算法.Song[1,2]等人提出了一种多层次的资源调度方案来优化 CPU、内存资源的分配,相应的设计了一个本地资源分配算法来为虚拟机动态再分配资源,以及一个全局调度算法来调整每个运行着的服务,控制资源在服务之间的流动,然而该调度算法依赖于具体应用,具有很大的局限性.Weng 等人[3]从理论角度分析了虚拟机监视器中异步 CPU 调度策略,并提出了一种混合的虚拟机调度框架.该框架主要解决多应用并发执行时物理 CPU 时间浪费问题,并没有考虑虚拟机的负载均衡.Zhou[4]等人认为迁移虚拟机的代价大于在物理机上为虚拟机重新分配资源的代价,于是提出了提前在物理机上进行资源调整,当负载达到物理机的阈值时再进行迁移,但相关资源调度算法并没有给出资源分配的具体额度,适应性不强.袁爱平等人[5]提出了一种多维资源协同聚合的虚拟机资源调度算法.该算法在分组遗传算法的基础上,采用模糊逻辑及基于资源利用率多维方差来设计适应度函数.该算法在一定程度上能够提高资源综合利用效率,但搜索解空间耗时过长,从而导致作业完成时间明显延长.Yuan 等人提出了 IPSO 虚拟机资源调度算法[6].该算法基于粒子群优化,主要考虑了服务器的计算能力,能够在一定程度上减少作业执行时间和改善系统的负载均衡,但由于只考虑了 CPU 资源,因而其适用性受到一定限制.Beloglazov 等人提出了应用于云数据中心的资源管理策略[7,8].该策略在保证不违反 SLA 的同时,利用虚拟机迁移技术不断把虚拟机整合到较少的服务器上,并关闭多余的空闲服务器来降低能耗.Kusi[9]等人研究了虚拟化 Web 集群的电能和性能之间的平衡问题,提出了限制预控制策略,其根据当前的 Web 请求负载,部署用于处理 Web 请求的虚拟机,在保证不违反 SLA 的同时使得电能消耗最少.Nakku 等人[10]提出了一种虚拟机能耗模型来估计每个虚拟机的能耗,以此为依据来提供计算资源,但该研究只考虑了处理器资源,并不考虑存储和网络能源消耗.上述工作[7−10]主要从节能角度出发来考虑资源的调度问题,而并不关注负载均衡对系统性能的影响.黄纬等人提出了一种基于 *K*-均值聚类的虚拟机资源调度算法[11].该算法利用虚拟机资源配置的相关性进行聚类,实现资源互补以提高虚拟机的稳定性,基于贪心算法进行虚拟机资源调度,但该算法会导致系统出现负载不均.刘东林等人[12]提出了一种基于分类挖掘的虚拟机资源调度模型,以解决资源碎片导致资源调度效率低的问题,但是没有考虑用户请求的动态变化,虚拟机资源间的耦合度和虚拟机各类资源的优先级.基于现有工作大都并不关注系统负载均衡问题,故本文基于 Xen 虚拟化平台提出了两种资源调度优化算法,即细粒度资源调度算法和粗粒度迁移调度算法,细粒度调度算法能够根据虚拟机当前的 CPU 负载情况动态修改权值和最大 CPU 时间片以增强高载虚拟机的性能,减少低载虚拟机的 CPU 资源浪费,通过对虚拟机内存使用率的监控和历史使用率信息对将来的内存利用情况进行预测,并结合气球驱动模型动态调整虚拟机内存大小,能够从 CPU 和内存资源两个方面实现对虚拟机资源的调度优化,避免不必要的迁移.对于无法用细粒度算法进行优化的情况,本文提出了一种基于虚拟机动态迁移的资源调度优化算法,即粗粒度迁移调度算法.该算法以迁移代价为依据来选择待迁移的虚拟机,并为待迁移虚拟机选择合适的物理机.该问题属于 NP 完全问题.本文结合粒子群优化算法,根据任务需求改进了粒子距离计算公式,提出了适合于资源调度的多目标评价函数,并通过适应度来评价解的品质.该算法具有搜索速度快、需要调整参数少、全局收敛能力强及鲁棒性好等优点.通过对两种资源调度优化算法进行实验,结果表明,这两种算法能够有效解决虚拟机资源调度优化问题,具有较高的性能和适用性.

1 细粒度资源调度优化算法

针对单个物理节点上虚拟机资源分配不合理问题,本文提出细粒度资源调度算法来优化 CPU 和内存资源的分配.该算法根据各个虚拟机的实际负载和需求,改进原有 Credit 算法,并利用 Xen 提供的 API 函数来为它们重新分配资源,从而能够动态调整虚拟机的物理资源,减少资源浪费,并保证系统的良好运行.本节将详细阐述 CPU 和内存资源重新分配的额度以及分配策略.

1.1 CPU资源调整

CPU 资源调整算法是在 Credit 调度算法的基础上,通过调整资源分配额度来进行优化的.下面具体给出调整额度的计算公式.

设 $nVcpu_i$ 表示第 i 台虚拟机的 VCPU 的个数,$i\in[1,V]$,V 表示虚拟机的个数,W_i 为虚拟机 i 的权重(weight),表示该虚拟机可占用 CPU 时间的比例(在 Credit 算法中其默认值为 256,有效值范围为[1,$2^{16}-1$]),Cap_i 表示虚拟机 i 可占用 CPU 时间的上限,该参数与 Credit 算法中的 Cap 值相同(Cap 为 0 表示不限制使用物理 CPU 的个数),f_i 为虚拟机 i 的主频.W_{total} 为物理机上所有虚拟机的权重之和,物理机的 CPU 核数为 K_{num},物理机的 CPU 主频为 f,则虚拟机的处理能力 Ca_i 为

$$\begin{cases} Ca_i = nVcpu_i \times \dfrac{W_i}{W_{total}} \times \dfrac{Cap_i}{K_{num} \times 100} \times \dfrac{f_i}{f}, & Cap_i \neq 0 \\ Ca_i = nVcpu_i \times \dfrac{W_i}{W_{total}} \times \dfrac{f_i}{f} & Cap_i = 0 \end{cases} \tag{1}$$

若用 Cu_i 表示虚拟机 i 的 CPU 利用率,Cu_e 表示进行 CPU 资源调整后期望的 CPU 利用率,Ca_i'表示调整后虚拟机 i 的处理能力,由于资源调整前后所处理的任务量不变,故下式成立:

$$Cu_i \times Ca_i = Cu_e \times Ca' \tag{2}$$

鉴于 Credit 算法中 Cap_i 值的特殊性,调整前后保持不变,仅通过调整权重 W_i 来进行优化,此时可以由式(1)和式(2)得到式(3):

$$W_i'/W_{\text{total}}' = Cu_i/Cu_e \times W_i/W_{\text{total}} \tag{3}$$

其中,W_i'为调整后的权重,由 $W_{\text{total}}' = W_{\text{total}}$,并令期望 CPU 利用率 Cu_e 为当前物理节点上所有虚拟机的 CPU 利用率期望,则 $Cu_e = \sum_{j=1}^{V}\left(Cu_j \times \dfrac{W_j}{W_{\text{total}}}\right)$,将其带入式(3),可得:

$$W_i' = \frac{W_{\text{total}} \times Cu_i \times W_i}{\sum_{j=1}^{V}\left(Cu_j \times W_j\right)} \tag{4}$$

引入虚拟机 i 的 CPU 利用率的预测值 Cp_i 并与当前 CPU 利用率进行加权作为 CPU 的将来 CPU 利用率,则 W_i 可用式(5)来表示:

$$W_i' = \frac{W_{\text{total}} \times W_i \times (\omega \times Cp_i + (1-\omega) \times Cu_i)}{\sum_{j=1}^{V}\left(Cu_j \times W_j\right)} \tag{5}$$

其中,ω表示对该预测值的信任程度,ω为 1 表示绝对信任该预测值,ω为 0 表示完全不信任该预测值.可以看出,只要得到 Cp_i,就可以计算出要调整的权重 W_i',从而实现资源调整,Cp_i 可以根据历史数据统计得出.

当$\forall i\in[2,V]$,$W_i'<2^{16}-1$ 时,表示所有权值可用,则可以为所有虚拟机调节权值;当$\exists i\in[2,V]$,$W_i'>2^{16}-1$ 时,则表示需要调节较多的 CPU 资源才能满足该虚拟机的需求,无法通过动态调整来解决此问题,需要将其宿主机设置为热点,然后通过虚拟机迁移调度算法来实现资源调整.算法 1 为具体的 CPU 资源调整优化算法.

算法 1. CPU 资源调整优化算法.

输入:CPU 利用率、当前分配的 Weight 权值、CPU 利用率预测值.

输出:是否触发迁移.

```
1    Bool Credit_CPU_Adjust(float [V], long M[V])
```

```
{Cu_i=updateVMCpuUsage();  //获取虚拟机的虚拟 cpu 利用率
 W_i=getVMCredit();  //获取 credit 调度算法中的 weight 值
 Cp_i=predict();  //获取虚拟机的 cpu 利用率的预测值
 For (i=1; i<V; i++)
        W_i' = (ω × Cp_i + (1 − ω) × Cu_i) × W_i };
    For (i=1; i<V; i++)
        If (W_i'>2^16−1) Return true;  //触发迁移
    For (i=1; i<V; i++)
       {W_i=W_i';  //为所有虚拟机重新分配 weight 值
         Return false;  }
}
```

1.2 内存资源调整优化

内存资源调整优化算法也是在 Credit 调度算法的基础上,通过修改资源分配额度来实现的,与 CPU 资源调整优化算法类似,下面具体给出相关的额度计算公式.

设 Mu_e 表示虚拟机的内存使用率上限,Mu_i 表示虚拟机 i 的当前内存利用率,M_i 表示虚拟机 i 的当前分配的内存大小,$i\in[1,V]$,V 表示该物理机上的正在运行的虚拟机的个数,i=1 时表示 Domain 0,由于 Domain 0 仅作为 Xen 的管理助手,一般情况下并无用户进程运行,且 Domain 0 在整个虚拟化系统中地位特殊,仅在创建、管理和配置其他虚拟机时涉及到内存变化,但这种变化一般都不大,所以这里不调节 Domain 0 的内存.设 M_T 为物理机的总物理内存大小,M_d 表示物理机的空闲内存容量,R 表示物理机的内存保留量(可用于为 Domain 0 应急内存或其他用途,也可以设定为 0 表示不保留内存),则有:

$$M_d = M_T - \sum_{i=1}^{v} M_i - R,\ i \in [1,V] \tag{6}$$

Φ表示高内存负载虚拟机集,这里将内存使用率高于给定的阈值 Mu_e 的虚拟机纳入到Φ集合中,即$\Phi=\{Mu_i|Mu_i>Mu_e, i\in[2,V]\}$,对于$\Phi$中的任一虚拟机 j,其对应虚拟机的期望内存大小为 W_j',则有:

$$(Mu_e - \tau) \times W_j' = (\omega \times Mp_j + (1-\omega) \times Mu_j) \times M_j \tag{7}$$

其中,$(Mu_e-\tau)$表示调整内存后的期望内存使用率,Mp_j 表示虚拟机 j 的内存使用率预测值,ω表示对该预测值的信任程度,当ω=1 表示绝对信任该预测值,ω=0 表示完全不信任该预测值.可以得到公式(8):

$$M_j' = \frac{\left(\omega \times Mp_j + (1-\omega) \times Mu_j\right) \times M_j}{(Mu_e - \tau)} \tag{8}$$

对$\forall i\in[2,V]$,按照公式(8)来计算各虚拟机期望内存大小,并重新设置各个虚拟机内存.算法 2 为具体的内存资源调整优化算法.

算法 2. 内存资源调整优化算法.

输入:内存使用率、已分配内存大小.

输出:是否触发迁移.

```
bool Memory_adjust(float Mu[V], long M[V])
{ M_d = M_T − Σ_{i=0}^{V−1} M_i − T;  //计算剩余内存(可用内存)大小
    For (i=0; i<V, i++){  //求出高载虚拟机集
        If (Mu_i> Mu_e) Φ[j++]=Mu_i;}
    For (i=1; i<j; i++)
        M_i' = (ω × Mp_i + (1 − ω) × Mu_i) × M_i / (Mu_e − τ);
```

```
7          If ( $\sum_{i=i}^{j} M'_i > M_d + \sum_{i=1}^{j} M'_i$ )
8              {Return false;}
9          else {
10             For (i=1; i<V; i++)
11                 $M'_i = \frac{(\omega \times Mp_i + (1-\omega) \times Mu_i) \times M_i}{(Mu_e - \tau)}$
12                 If ( $\sum_{i=2}^{V} M'_i < M_d + \sum_{i=2}^{V} M'_i$ )
13                 {For (j=1; j<V; j++)
14                     If (M'j <Mj) Mj=M'j;
15                   For (j=1; j<V; j++) Mj=M'j;
16                     Return false;
17                 }
18                 Else {Return true;}
19         }
20     }
```

当 $\sum_{i=2}^{V} M'_i < M_d + \sum_{i=2}^{V} M_i$ 时,表示通过调整所有虚拟机的内存能够满足高载虚拟的需求,此时按照 M'_i 为虚拟机 i 重新分配内存即可,这里需要先回收内存才能为高载虚拟机增加内存,所以调整过程需要先为 $M'_i<M_i$ 的虚拟机调整,从这些虚拟机中回收内存,然后再为 $M'_i>M_i$ 的虚拟机调整,为它们增加内存.

当 $\sum_{i=2}^{V} M'_i > M_d + \sum_{i=2}^{V} M_i$ 时,说明该物理机上的内存无法满足高载虚拟机的要求,此时只能通过迁移调度方案来调整物理机的负载.

2 粗粒度资源调度优化算法

在由许多虚拟节点组成的云计算集群中,当各个物理机负载出现严重不均衡,或集群中出现热点时,通过迁移高载物理机(热点)上的部分虚拟机到低载的物理机上,以达到整个虚拟化集群负载均衡的目的.为此,本文结合粒子群算法提出了基于迁移的资源调度优化算法.该算法主要由待迁移虚拟机的选择和基于粒子群的迁移调度算法两部分构成,下面分别进行详细论述.

2.1 待迁移虚拟机的选择

由于动态迁移过程中仅需要迁移 CPU 状态、内存及网络状态,在网络带宽相同的情况下,其迁移开销可以用迁移过程中需要传输的数据量衡量,通常迁移开销最大的是虚拟机内存部分的迁移.本文主要基于以下两种情况来选取待迁移的虚拟机.

(1) 当一个物理机本身负载很高,并超出一定的限度时,会严重影响系统的性能和稳定性,也无法通过节点内部资源的重新分配来满足高负载虚拟机的性能要求,此时就必须通过迁移虚拟机来降低其负载.

(2) 当集群中各物理机的负载都不高时,出于节能的考虑,可以将部分物理机上的虚拟机全部迁移到其他物理机上,然后将这部分物理机切换到待机模式或通过 CPU 智能降频等手段来达到节能的目的.

选择待迁移虚拟机时,应综合考虑待迁移虚拟机的内存大小和虚拟机的负载,迁移内存大的虚拟机迁移开销大,迁移负载高的虚拟机对物理机的负载影响大,所以本文的策略是尽量选择负载高且内存小的虚拟机作为待迁移的对象.可用如下公式来量化迁移代价,设 Uc_i 表示虚拟机 i 的物理 CPU 利用率,M_i 表示虚拟机 i 的内存大小,Um_i 表示虚拟机 i 的内存利用率,W_i 表示虚拟机 i 的内存使用量与负载的比值,则 $W_i=(Um_i\times M_i/(Uc_i+Um_i).W_i$ 将作为量化虚拟机 i 迁移的代价的重要因素.在选择待迁移虚拟机时,管理节点从负载较高的物理机上选择具有最小 W_i 的 k 个虚拟机作为迁移对象,k 可以定义为选出 k 个虚拟机后的物理机的负载最接近集群负载的平均值或物理机负载达到负载阈值.所有待迁移虚拟机按照负载大小从由高到低进行排序,并将以该次序产生待迁

移虚拟机集合,并作为粒子群算法的个体(粒子)的结构.

2.2 基于粒子群的迁移调度算法

为了寻找出待迁移虚拟机和物理机之间最佳映射关系,我们基于粒子群优化算法来解决此问题,从 CPU 和内存两个方面综合考虑,找到最佳的映射关系.为此本文首先提出了相应的多目标适应度函数.

(1) CPU 负载均衡度.CPU 负载均衡度表示物理机的 CPU 负载的均衡程度,用整个物理集群 CPU 利用率(负载)的标准差来表示.当前粒子所代表的集群状态中的各个物理机的 CPU 的利用率,由于粒子代表的是迁移完成后的状态,所以需要对迁移完成后的物理机 CPU 利用率进行预测,由于迁移前后虚拟机的处理能力不变,则有如下等式:

$$VM'_{PCPU}=\frac{K_{num1}\times f_1}{K_{num2}\times f_2}\times VM_{PCPU} \tag{9}$$

其中,K_{num1} 和 f_1 分别表示迁移前虚拟机所处的物理机的 CPU 核心数和主频,K_{num2} 和 f_2 分别表示迁移后虚拟机所处的物理机的 CPU 核心数和主频,VM_{PCPU} 表示迁移前虚拟机的物理 CPU 利用率,VM'_{PCPU} 表示迁移后虚拟机的物理 CPU 利用率,所以,迁移后目标宿主机的 CPU 利用率应该为迁移前的该物理机的 CPU 利用率与 VM'_{PCPU} 之和.设 E_{cpu} 为迁移完成后的所有物理机的 CPU 利用率的期望,则有:

$$E_{cpu}=\frac{1}{P}\sum_{i=1}^{P}Cu_i \tag{10}$$

其中,Cu_i 表示第 i 台物理机的 CPU 利用率,P 表示集群中的物理机的台数.所以,物理机 CPU 负载均衡度为

$$S_P=\sqrt{\frac{1}{P}\sum_{i=1}^{P}(Cu_i-E_{cpu})^2} \tag{11}$$

显然,CPU 负载均衡度 S_P 的值越小表明该粒子所代表的迁移结果越好.设 S_P'表示迁移前 CPU 负载均衡度,S_{CPU}作为在物理机 CPU 利用率这个适应度函数上的结束状态,当 $S_P<S_P'-S_{CPU}$时则在该适应度函数上达到了可结束状态.

(2) 内存余量均衡度.内存余量均衡度用物理机内存剩余率的标准差来表示.物理机内存余量(*Lm*)可以定义为物理内存减去该物理机上当前虚拟机获得的内存的差值,即

$$Lm_i = M_{Ti} - \sum_{j=1}^{V} M_{ij} - R_i \tag{12}$$

其中,M_{Ti} 表示第 i 台物理机的物理内存总量,M_{ij} 表示第 i 台物理机上的第 j 台虚拟机的内存,R_i 表示物理机 i 的内存保留量,V 为该物理机上的虚拟机个数.物理机 i 的内存剩余率为 $Lv_i=\frac{Lm_i}{M_{Ti}}$, 集群中物理机内存剩余率的期望为 $E_m=\frac{1}{P}\sum_{i=1}^{P}Lv_i$, 则物理内存余量均衡度为

$$S_M=\sqrt{\frac{1}{P}\sum_{i=1}^{P}(Lv_i-E_m)^2} \tag{13}$$

可以看出,内存余量均衡度 S_M 越小表明该粒子所代表的迁移结果越好.设 S_M'表示迁移前的内存余量均衡度,S_{MEM}作为物理机内存余量率上的结束状态,即当 $S_M<S_M'-S_{MEM}$时,在该适应度函数上满足结束状态.

使用多目标适应度函数后,粒子的历史最优解(*pBest*)也发生了变化,本文利用多目标函数加权求和的方式来决定 *pBest*:

a) 第 j 个粒子的第 i 个目标函数(评价函数)值为 P_{ij}(其中,j=1,2,3,…,N;N 为粒子个数,i=1,2),则粒子 j 的当前性能 pB_j'为

$$pB_j'=W_1\times P_{1j}+W_2\times P_{2j} \tag{14}$$

其中,W_i 表示第 i 个目标函数的权值.设该粒子的历史最优解为 $pBest_j'$,将 pB_j'和 $pBest_j'$进行比较,$pBest_j=\min\{pB_j',pBest_j'\}$.

b) *gBest* 依旧按照标准的粒子群算法进行更新:

$$gBest=\min\{pBest_j\} \tag{15}$$

粒子群种群的初始化作为算法运行的第 1 步,无疑是很重要的,设定 N 为种群大小,即粒子的个数.由于距离是以物理机之间的跳步数确定,所以,本文限定粒子在每一维上的速度上限都为 $V_{\max}$.种群中的各个粒子的初始位置一般都是随机产生,初始速度设定为 0,方向以物理机编号从小到大为正向.当某个粒子在所有的适应度函数上都"达到"结束状态时,粒子群算法即可结束.当算法运行了多代后都未能找到满足结束状态的解时,则需要考虑通过限定算法的迭代次数($ITR_{\max}$)来结束算法,即当算法迭代的次数达到 $ITR_{\max}$ 时,强行结束算法,并将结束前的 *gBest* 对应的粒子作为解.

在迭代的过程中,粒子的位置是不断变化的,每个粒子在下一时刻移动的速度将由该粒子的历史最优解和全局最优解共同决定,在标准的粒子群算法中还加入了该粒子的惯性,即粒子上一时刻的速度也决定粒子将来的速度.粒子 i 在第 d 维上的速度和位置的更新按如下公式执行:

$$\begin{cases} Vid(t+1)=w\times Vid(t)+c1\times rand1\times[Pid-Xid(t)]+c2\times rand2\times[pgd-Xid(t)] \\ Xid(t+1)\ =\ Xid(t)\ +\ Vid(t+1) \end{cases} \tag{16}$$

其中,w 表示惯性权重,$V_{id}(t)$表示 t 时刻粒子 i 在第 d 维上的速度,c_1 和 c_2 分别表示粒子 i 在自身经验和全局经验上的学习常数,$rand_1$ 和 $rand_2$ 都表示 0~1 之间的随机数,P_{id} 表示粒子 i 的历史最优解在第 d 维上的位置,P_{gd} 表示粒子群全局最优解在第 d 维上的位置,$X_{id}(t)$表示 t 时刻粒子 i 在第 d 维上的位置,$[P_{id}-X_{id}(t)]$表示粒子 i 在第 d 维上到其历史最优值的距离(跳步数),$[P_{gd}-X_{id}(t)]$同理.基于粒子群的迁移调度优化算法见算法 3.

算法 3. 粒子群迁移调度优化算法.

输入:待迁移虚拟机集,物理机集.

输出:迁移映射方案.

```
1   bool PSO(VM vms[],PM pms){
2   for (int i=0; i<N; i++){//初始化粒子 i
3         for (int j=0; j<V; j++){
4            Part[i][j]=random(1,P);}}
5   for (itr=0; itr<ITR_max; itr++)   //ITR_max 最大迭代次数
6         { for (i=0; i<N; i++)   //更新粒子的历史最优解
7          { pi= fitness(Part[i]);   //计算 i 的适应度函数
8              if(pi<pBest[i]) {   //历史最优粒子的位置或状态
9                 pB[i]=Part[i];
10                pBest[i]=pi;}}
11             for (i =0; i<N; i++)   //更新全局最优粒子
12               If (pBest[i]<gBest) {
13        gBest=pBest[i];   //全局最优粒子的适应度
14        gB=pB[i]; }   //全局最优粒子的位置或状态
15        if (satisfy(gB)){   //如果达到了预定的结束状态
16                  return gB;}   //跳出循环,结束算法
17        for (i=0; i<N; i++)   //更新粒子速度和位置
18           { for (int j=0; j<P; j++) {
19  v[i][j]=w×v[i][j]+c1×rand()×((pB[i][j]-Part[i][j]+P)%P)+c2×rand()×((gB[j]-Part[i][j]+P)%P);
20             if(v[i][j]>V_max)   v[i][j]=V_max;
21             Part[i][j]=(Part[i][j]+v[i][j])%P;}
22               }}
23        return gB;
```

24 }

3 性能评价与分析

为了验证两种资源动态优化算法的性能,本文构建了实验测试平台.实验测试平台共由 10 台 IBM X3650 M4 服务器和 1 台 IBM V7000 SAN 存储组成,其中,SAN 存储为整个平台提供共享存储,所有虚拟机的磁盘镜像存放于 SAN 存储上.其余 10 台服务器都安装 Centos 6.4 和 Xen 4.1.2 虚拟机监视器、相关算法模块以及其他测试工具软件,进行了性能测试.下面分别给出测试结果和详细分析.

3.1 细粒度动态优化算法

(1) CPU 资源调度优化.为了验证 CPU 资源调整优化的效果,我们在两个物理节点 A 和 B 上分别启动 3 个相同配置的虚拟机,其中,分别在 A 和 B 的第 1 个虚拟机 HA-H001 和 HB-h001 虚拟机上同时运行 Super PI 测试程序来模拟 CPU 密集型负载,而各自的第 2 和第 3 号虚拟机仅保持开机状态,而不运行任何任务,同时在物理机 A 上运行本文所提出的 CPU 资源优化算法(改进的 Credit 算法),物理机 B 上运行原有 Xen 资源调度算法(Credit 算法).图 1 是两个虚拟机 HA-h001 和虚拟机 HB-h001 运行时 CPU 利用率的变化情况.该图中将取样频率设定为 200ms 一次,该图中进行了 800 次 CPU 利用率的取样.

从图中看出,随着运行负载的增加,两个虚拟机的 CPU 利用率在缓慢增加,在 550 次取样之前,两台虚拟机的 CPU 资源利用率的变化趋势基本一致,而在此之后,由于 CPU 的利用率超过了 70%的阈值,而物理机 A 上运行的本文提出的改进 Credit 算法开始增加虚拟机 HA-h001 的 CPU 资源,从而使得其利用率下降,一直下降到 40%~50%的合理区间,之后就基本稳定在该区间运行.而物理机 B 上的原有 Credit 算法由于动态调整资源的能力,因而虚拟机 HB-h001 的 CPU 利用率一直保持在 70%上下浮动,可以看出,本文提出的 CPU 资源动态优化算法能够在单物理机上有效的为高载虚拟机增加更多的资源,以保证其资源利用率保持在期望范围内.

(2) 内存资源调度优化.在上述硬件环境下,为了验证内存资源优化的效果,我们在物理节点 A 和 B 上的第 1 台虚拟机上运行相同的内存密集型任务,而第 2、3 台虚拟机则仅保持开机状态,而不运行任何任务.图 2 是两个同时运行内存密集型任务的虚拟机的内存利用率的对比图.其中 HA-p001 表示运行优化算法的物理节点(Host A)上的虚拟机的内存利用率,而 HB-p001 表示没有运行优化算法的物理节点(Host B)上的虚拟机的内存利用率,该图共进行了 120 次内存利用率的取样.从图中可以看出,当虚拟机的内存利用率超出 70%时,物理机 A 上的优化算法开始增加虚拟机 HA-p001 上的内存资源,使得内存利用率保持在 40%~50%的期望区间,物理机 B 上运行的 credit 算法由于不具有资源动态调整能力,因此其虚拟机 HB-p001 的内存利用率一直处于 70%以上,直到任务完成为止.

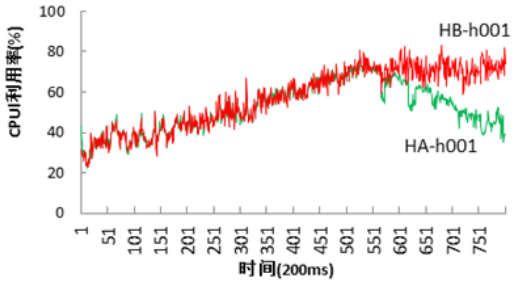


Fig.1 Comparison of scheduling optimization of CPU resource

图 1 CPU 资源调度优化对比

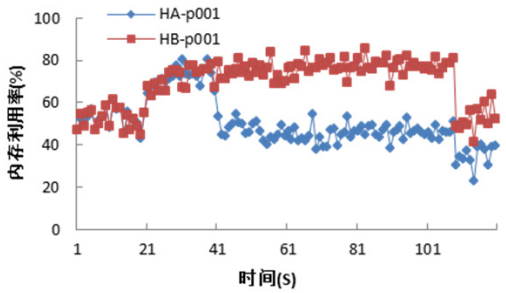


Fig.2 Comparison of scheduling optimization of memory resource

图 2 内存资源优化对比

本文针对 CPU 和内存优化调整做了大量实验,由于篇幅有限,仅给出其中两个简单实验.但从中可以看出,与原有 credit 调度算法相比,由于增加了 CPU 和内存自动调整模块,因而我们提出的改进算法确实能够根据虚

拟机应用需求来动态地调整单个物理机上的虚拟机的资源,从而避免了必要的迁移和系统的宕机,具有较好的性能和适用性.

3.2 粗粒度动态优化算法

针对云环境下的多物理机集群系统的负载均衡问题,本文提出了基于改进粒子群的虚拟机资源调度优化算法,以生成迁移调度策略,按照这一策略实现虚拟机迁移,进而达到集群物理机的负载均衡.为了验证该算法的性能,我们在搭建的由 10 台物理机构成的云平台上,创建了相应的虚拟机.由于本算法主要是求解最佳的实现负载均衡迁移方案,并且综合考虑 CPU 和内存资源,而现有的算法主要解决调度问题,并且仅考虑 CPU 资源,因而不具有可比性.下面从 3 个方面来对该算法的性能进行分析.

(1) 适应度函数权重影响.在改进的粒子群算法中,本文主要从 CPU 和内存负载两方面来考虑迁移优化策略,因此将多适应度函数通过加权求和作为总适应度函数来控制产生迁移策略,因此适应度函数权重的选择对算法的性能具有重要影响.为此,本文设计在 5 个物理节点上存在热源,其他 5 个物理节点作为迁移的目的宿主机,并且其中有一台启动少量虚拟机占用了少量内存,但运行着 CPU 密集型任务;而另一台目标宿主机上同时运行着较多的虚拟机,已分配了较多的内存,但所有虚拟机无任务运行.在总适应度函数中内存余量率的权重以 0.1 为间隔,从 0 到 1 变化的过程中,迁移的执行时间如图 3 所示,分别为有 10 台、7 台和 5 台虚拟机进行迁移时的迁移执行时间变化情况.从图中可以看出,当内存余量均衡度的权重取值为 0.4(CPU 负载均衡度权重取值为 0.6)时,迁移的执行时间最少,可见适应度函数的权重取 0.4 左右为宜.

(2) 确定惯性权重 W 和种群大小 N.为了确定惯性权重 W 和准确大小 N,本文对于惯性权重 w 取值从 0.4 开始,以 0.1 为间隔增加直到 1.4 为止,粒子种群大小 N 取值以 10 为间隔,从 10 到 100 变化,以循环代数 1000 或均衡度改善 10 以上作为算法结束条件,图 4 是寻优次数的变化情况.从中可以看出,w 在取值为 1 时,从满足结束状态而退出的寻优次数最多,当 w 的值大于 1.2 以后,算法求解的性能开始变差,并随着 w 的增加,求解性能出现持续下降.同时可以看出,在种群数量 N>30 之后的曲线都比较接近,成功求解的次数都在 900 次以上.因此,粒子群的种群大小设置为 30,粒子群算法的惯性权值 W 设定为 1.0 为宜.

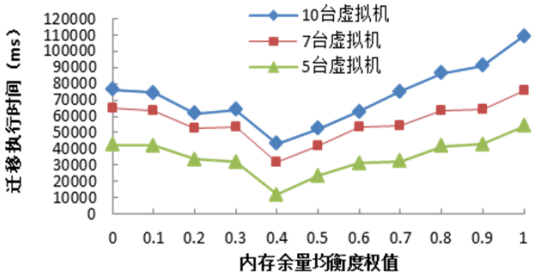


Fig.3 Varying of migration execution time with the weight of the memory remaining equilibrium degree

图 3 迁移执行时间随内存余量均衡度权值变化情况

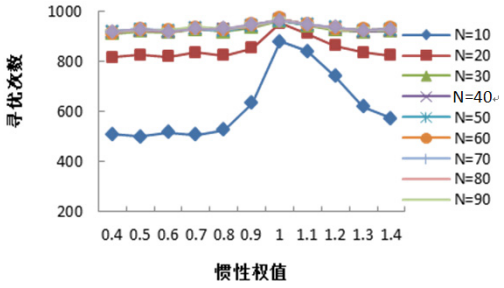


Fig.4 Varying of optimization times with the inertia weight

图 4 寻优次数随惯性权重的变化情况

(3) 迁移前后负载均衡度变化.为了粒子群迁移优化算法的有效性,本文在 5 台物理机上创建了不同的虚拟机,并运行相应的负载.表 1 给出了各个虚拟机在触发迁移调度之前,各个运行的虚拟机资源利用情况,其中,HA-h00x 表示 Host A 上的第 x 台正在运行的虚拟机,VCPU usage 表示该虚拟机的 CPU 利用率,PCPU usage 表示该虚拟机的物理 CPU 的利用率,Memory 表示该虚拟机当前占有的内存大小,单位为 GB.

根据集群中虚拟机资源状态,迁移调度器通过待迁移虚拟机选择算法,从集群中各个负载较高的物理节点上选择出合适虚拟机进行迁移,并分别选取 HA-h010,HA-h001,HA-h002 和 HB-h001 共 4 台虚拟机作为待迁移虚拟机.并根据粒子群映射算法输出结果[5,5,4,3]进行迁移,迁移的顺序为 HA-h010 与 HB-h001 同时启动,并分别被迁移到物理节点 host E 和 Host C 上,HA-001 和 HA-h002 随后被分别迁移到 Host D 和 Host C 上,迁移后集

群的状态见表 2.

Table 1 Cluster status before migration

表 1 迁移前集群状态

虚拟机编号	VCPU usage (%)	PCPU usage (%)	Mem usage (%)	Memory (G)
HA-h001	0.21	0.081	0.31	1
HA-h002	0.18	0.118	0.29	1
HA-h003	0.09	0.03	0.13	2
HA-h004	0.31	0.131	0.47	2
HA-h005	0.16	0.03	0.24	2
HA-h006	0.43	0.113	0.59	2
HA-h007	0.29	0.079	0.41	4
HA-h008	0.24	0.024	0.35	4
HA-h009	0.27	0.027	0.41	4
HA-h010	**0.73**	0.23	0.87	4
Host A	–	0.863	–	26
HB-h001	0.28	0.056	0.31	1
HB-h002	0.37	0.074	0.45	1
HB-h003	0.19	0.038	0.28	2
HB-h004	0.19	0.138	0.27	2
HB-h005	0.21	0.142	0.34	4
Host B	–	0.448	–	10
HC-h001	0.17	0.056	0.27	1
HC-h002	0.08	0.03	0.17	2
HC-h003	0.11	0.1	0.19	4
Host C	–	0.186	–	7
HD-h001	0.18	0.09	0.15	2
HD-h002	0.14	0.07	0.38	4
Host D	–	0.16	–	6
HE-h001	0.17	0.017	0.32	2
Host E	–	0.017	–	2

Table 2 Cluster status after migration

表 2 迁移后集群状态

虚拟机编号	VCPU usage (%)	PCPU usage (%)	Mem usage (%)	Memory (G)
HA-h003	0.09	0.03	0.13	2
HA-h004	0.31	0.131	0.47	2
HA-h005	0.16	0.03	0.24	2
HA-h006	0.43	0.113	0.59	2
HA-h007	0.29	0.079	0.41	4
HA-h008	0.24	0.024	0.35	4
HA-h009	0.27	0.027	0.41	4
Host A	–	0.434	–	20
HB-h002	0.37	0.074	0.45	1
HB-h003	0.19	0.038	0.28	2
HB-h004	0.19	0.138	0.27	2
HB-h005	0.21	0.142	0.34	4
Host B	–	0.392	–	9
HC-h001	0.17	0.056	0.27	1
HC-h002	0.08	0.03	0.17	2
HC-h003	0.11	0.1	0.19	4
HB-h001	0.28	0.056	0.31	1
Host C	–	0.242	–	8
HD-h001	0.18	0.09	0.15	2
HD-h002	0.14	0.07	0.38	4
HA-h002	0.18	0.118	0.29	1
Host D	–	0.278	–	7
HE-h001	0.17	0.017	0.32	2
HA-h010	0.73	0.23	0.87	4
HA-h001	0.21	0.081	0.31	1
Host E	–	0.328	–	7

在整个自动迁移过程中,CPU 均衡度和内存的均衡度的变化情况如图 5 所示.从图中可以看出,4 台虚拟机的迁移共用时 169s,其中 HB-h001 最先于 26s 时完成迁移,由于 HA-h010 负载高,内存大,所以完成时间较慢,用

时 109s,而后虚拟机 HA-h001,HA-h002 先后分别于 139s 和 169s 完成了迁移.在迁移过程中,集群的 CPU 负载和内存负载均衡度不断减少,分别从最初的 29.85 和 27.52 变为最终的 7.07 和 15.49,这说明负载越来越均衡.这说明本文提出的算法是有效的.

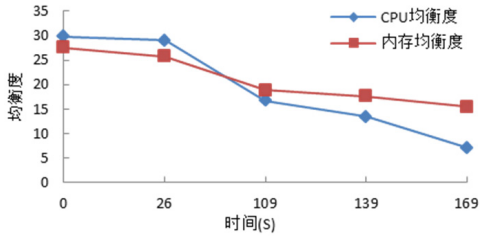


Fig.5 Varying of CPU and memory equilibrium degree in migration execution course

图 5 迁移执行过程中 CPU 和内存均衡度变化

4 结 论

本文主要对 Xen 虚拟机资源调度优化技术进行了较为深入的研究,提出了两种粒度的资源动态优化算法.细粒度的资源调度算法可以解决单个物理机上虚拟机资源分配不均的问题;粗粒度的基于粒子群的迁移调度算法可以在无法通过单个物理机局部调整的情况下,把部分虚拟机迁移到其他物理机上.这两种算法相互配合,能够进行虚拟机资源的动态优化,实现负载均衡.通过大量的实验分析了进行资源动态优化前后的资源利用情况,实验结果表明,与 Xen 原有的资源调度算法相比,本文所提出的两种算法具有较好的优化效果,能够在一定程度上解决云计算集群中虚拟机的负载均衡问题,从而避免负载不均所引起的性能瓶颈和宕机故障,下一步将结合节能和服务质量对算法的性能做进一步的优化完善.

References:

[1] Song Y, Li YQ, Wang H, Zhang YF, Feng BQ, Zang HY, Sun YZ. A service-oriented priority-based resource scheduling scheme for virtualized utility computing. In: Proc. of the High Performance Computing Conf. 2008. 220−231.

[2] Song Y, Wang H, Li YQ, Feng BQ, Sun YZ. Multi-Tiered on-demand resource scheduling for VM-based data center. In: Proc. of the Int'l Symp. on Cluster, Cloud and Grid Computing. 2009. 148−155.

[3] Weng CL, Wang ZG, Li ML, Lu XD. The hybrid scheduling framework for virtual machine systems. In: Proc. of the 2009 ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual execution environments. ACM, 2009. 111−120.

[4] Zhou WY, Yang SB, Fang J, Niu XL, Sun H. VMCTune: A load balancing scheme for virtual machine cluster using dynamic resource allocation. In: Proc. of the GCC 2010. 2010. 81−86.

[5] Yuan AP, Wang CJ. Virtual machine deployment strategy based on improved genetic algorithm in cloud computing environment. Journal of Computer Applications, 2014,34(2):357−359 (in Chinese with English abstract).

[6] Yuan H, Li CB, Du MK. Optimal virtual machine resources scheduling based on improved particle swarm optimization in cloud computing. Journal of Software, 2014,9(3):705−708.

[7] Beloglazov A, Buyya R. Energy efficient resource management in virtualized cloud data centers. In: Proc. of the 10th IEEE/ACM Int'l Conf. on Cluster, Cloud and Grid Computing.Washington: IEEE Computer Society, 2010. 826−831. [doi: 10.1109/CCGRID.2010.46]

[8] Beloglazov A, Abawajy J, Buyya R. Energy-Aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems, 2011,28(5):755−768. [doi: 10.1016/j.future.2011.04.017]

[9] Kusi CD, Kephart JO, Hanson JE, Kandasamy N, Jiang GF. Power and performance management of virtualized computing environments via look ahead control. Cluster Computing, 2009,12(1):1−15.

[10] Nakku K, Jungwook C, Euiseong S. Energy-Credit scheduler: An energy-aware virtual machine scheduler for cloud systems. Future Generation Computer Systems, 2014,32:128−137. [doi: 10.1016/j.future.2012.05.019]

[11] Huang W, Wen ZP, Chen C. Virtual machine scheduling algorithm based on *K*-means clustering in cloud computing. Journal of Nanjing University of Science and Technology, 2013,37(6):807−812 (in Chinese with English abstract).

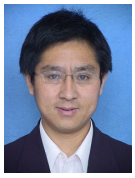
[12] Liu DL, Chen HB. The research of virtual machine resources scheduling based on CDVRS in cloud computing. Journal of Anhui University, 2013,37(4):40−45 (in Chinese with English abstract).

附中文参考文献:

[5] 袁爱平,万灿军.云环境下基于改进遗传算法的虚拟机调度策略略.计算机应用,2014,34(2):357−359.

[11] 黄纬,温志萍,程初.云计算中基于 *K*-均值聚类的虚拟机资源调度算法研究.南京理工大学学报,2013,37(6):807−812.

[12] 刘东林,陈宏滨.基于 CDVRS 的虚拟机资源调度策略研究.安徽大学学报,2013,37(4):40−45.



乔百友(1970－),男,甘肃礼县人,博士,副教授,主要研究领域为虚拟化,云计算技术.
E-mail: qiaobaiyou@ise.neu.edu.cn

王虹(1989－),女,硕士生,主要研究领域为云数据管理技术.
E-mail: wh_neu@163.com

蔡仁翰(1987－),男,助理工程师,主要研究领域为数字海洋,云计算技术.
E-mail: cairenhan@126.com

陈洋(1990－),男,硕士生,主要研究领域为资源虚拟化,云数据管理.
E-mail: chenyang_1010@126.com

陈东海(1988－),男,硕士生,主要研究领域为虚拟化技术.
E-mail: cdh_cjx@163.com

王国仁(1966－),男,博士,教授,博士生导师,主要研究领域为数据库技术,云计算技术,不确定数据管理技术.
E-mail: wanggr@mail.neu.edu.cn