软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(9):2193−2205 [doi: 10.3724/SP.J.1001.2011.04056]

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

一种面向虚拟化数字中心资源按需重配置方法*

米海波[1+], 王怀民[1], 尹 刚[1], 史殿习[1], 周扬帆[2], 袁 霖[1,3]

[1](国防科学技术大学 并行与分布处理国家重点实验室,湖南 长沙 410073)

[2](香港中文大学 计算机科学与工程学系,香港)

[3](解放军信息工程大学 电子技术学院,河南 郑州 450004)

Resource On-Demand Reconfiguration Method for Virtualized Data Centers

MI Hai-Bo[1+], WANG Huai-Min[1], YIN Gang[1], SHI Dian-Xi[1], ZHOU Yang-Fan[2], YUAN Lin[1,3]

[1](National Key Laboratory of Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China)

[2](Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong)

[3](School of Electronic Technology, Information Engineering University, Zhengzhou 450004, China)

+ Corresponding author: E-mail: rainmhb@gmail.com

Mi HB, Wang HM, Yin G, Shi DX, Zhou YF, Yuan L. Resource on-demand reconfiguration method for virtualized data centers. *Journal of Software*, 2011,22(9):2193−2205. http://www.jos.org.cn/1000-9825/4056.htm

Abstract: This paper proposes a resource on-demand approach for Web applications, which can efficiently online reconfigure clusters in response to time-varying resource requirements. It can also dynamically decide the number of running nodes and virtual machines deployed on them. It first predicts the future workloads of the applications with Brown's quadratic exponential smoothing method to make reconfiguration catch up with demands. Next, it adopts a genetic algorithm to parallel find the optimal reconfiguration policy. Experimental results demonstrate the approach can online adapt the cluster resource according to the change of requirement, increase the cluster resource utilization and greatly reduce power consumption.

Key words: data center; virtualization technology; online reconfiguration; genetic algorithm; power consumption

摘 要: 面向 Web 应用,提出一种动态资源按需配置方法,能够根据不断变化的资源需求以在线方式高效地重配置集群,实时地确定集群当前节点运行数量及其上部署的虚拟机类型.该方法基于布尔二次指数平滑法预测用户请求,有效避免了配置结果落后于资源请求;基于遗传算法并行化搜索配置空间,快速发现合理配置.实验结果表明,该方法能够根据需求变化高效地在线调整系统资源配置,并可有效提高集群资源利用率,显著降低了系统能耗.

关键词: 数据中心;虚拟化技术;在线重配置;遗传算法;能耗

中图法分类号: TP393 **文献标识码**: A

当前,能源开支已成为制约数据中心规模和效益的关键因素[1].据麦肯锡咨询 2008 年调查显示,全球数据中

* 基金项目: 国家自然科学基金(90818028, 60903043); 国家高技术研究发展计划(863)(2007AA010301); 国家杰出青年科学基金(60625203)

收稿时间: 2010-08-10; 定稿时间: 2011-04-28

心的耗电总量已在工业用电中占据第 4 位[2].截止 2008 年底,50%的数据中心面临能源不足问题[1].服务器(以下简称为节点)资源利用率低是数据中心能耗巨大的重要原因之一[3].据文献[2]统计,目前全球数据中心节点资源的平均利用率远低于期望值,如图 1 所示.如何提高资源利用率成为数据中心控制能耗必须解决的问题.虚拟化技术支持多个逻辑上独立的应用共享同一节点的物理资源[4],为提高节点利用率提供了可行的解决方案.但是,当前大多数数据中心往往根据虚拟机对资源的峰值需求来决定集群的配置方案,而峰值需求远大于常态需求,所以仍难以有效提高资源利用率.图 2 所示的是 1998 年世界杯官方网站 6 月 22 日的访问请求分布[5],其峰值需求约为常态需求的 10 倍.

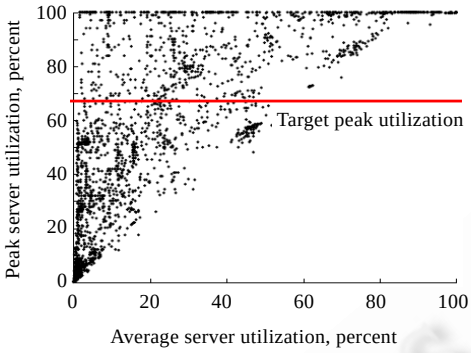


Fig.1 Resource utilization of data enter nodes

图 1 数据中心节点资源利用率

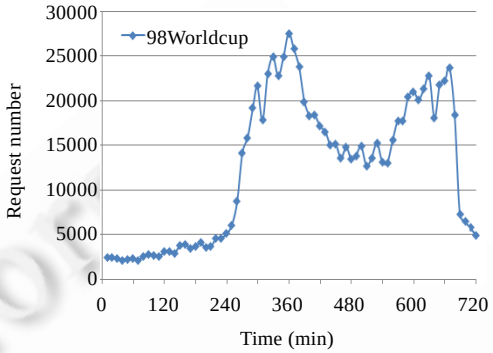


Fig.2 User request distribution of World Cup 98

图 2 1998 世界杯用户访问分布

一种理想的方法是,根据不断变化的需求和环境条件,如请求的类型与数量、节点负载情况、资源利用率等,对集群中的虚拟机进行动态配置(动态地调整节点运行数量及其上部署的虚拟机类型),在满足用户服务质量的前提下提高集群的资源利用率.目前,有很多研究试图解决数据中心的资源动态配置问题[3,4,6−15],然而大多数算法扩展性较差,仅适用于小规模集群.如文献[9]提出的算法,在 5 个节点的条件下,计算一次配置的时间小于 10s;而节点数为 15 时,计算时间接近 30m.另外,大多数研究工作仅根据当前时刻点的请求计算调整策略,忽视节点调整的时间开销,致使调整结果滞后于请求变化.

针对上述问题,本文面向虚拟化的数据中心,以 Web 应用为背景,提出一种集群资源按需动态配置方法.该方法基于布尔二次指数平滑法预测用户请求,能够有效避免重配置结果滞后于请求变化;基于遗传算法并行搜索配置空间,能够快速搜索出最优配置.实验结果表明,该方法能够根据需求的动态变化和节点的负载情况快速调整集群中节点运行的数量与其上部署的虚拟机类型,能够有效提高集群利用率,降低能耗.

本文第 1 节介绍自配置框架结构.第 2 节阐述问题模型,并描述相关算法.第 3 节介绍实验环境、设计细节和结果分析.第 4 节介绍相关工作.最后对全文进行总结.

1 自配置框架

在由多个节点构成的虚拟化集群中,假设存在 N 种 Web 应用 $\{A_i\}_{i=1}^N$,则对应 N 种虚拟机 $\{V_i\}_{i=1}^N$.设 A_i 运行在 V_i 上,每一种 V_i 在集群中的副本为 $\{C_{ij}\}_{j=1}^M$,M 为集群节点数目.C_{ij} 值域为{0,1},当 C_{ij}=1 时表示节点 j 启动 V_i;相反,当 C_{ij}=0 时表示未启动 V_i.我们将以下过程定义为自配置:在保证用户访问 QoS 的前提下,数据中心根据 $\{V_i\}_{i=1}^N$ 对节点资源的需求变化动态地调整 $\{C_{ij}\}_{i,j=1}^M$,关闭不必要开启的节点,提高集群资源利用率.

本文采用的自配置框架如图 3 所示.前端有 3 个核心构件组成:请求分发器、虚拟机管理器(virtual machine manager,简称 VMM)和重配置策略生成器(reconfiguration policy generator,简称 RPG).当集群中有新的虚拟机部署时,前端负责为其分配一个新的 VMM;请求分发器负责将对虚拟机的请求分发到对应的 VMM.由于重配置需要一定的时间,为了保证重配置之后的结构适应资源的需要,各 VMM 通过请求预测模块(request forecast

module)预测虚拟机对节点资源需求的变化趋势(在 Web 应用中,主要由用户对应用请求的变化确定),然后 VMM 将预测结果发给 RPG.RPG 根据预测值使用重配置搜索模块(reconfiguration searching module)从各种可能的配置空间中找到最优配置(虚拟机的副本数、位置和各自接收请求数),并将其返回给 VMM.VMM 根据此信息启动/关闭所管理的虚拟机.最后,VMM 根据配置信息进一步分发用户的请求到特定的节点.

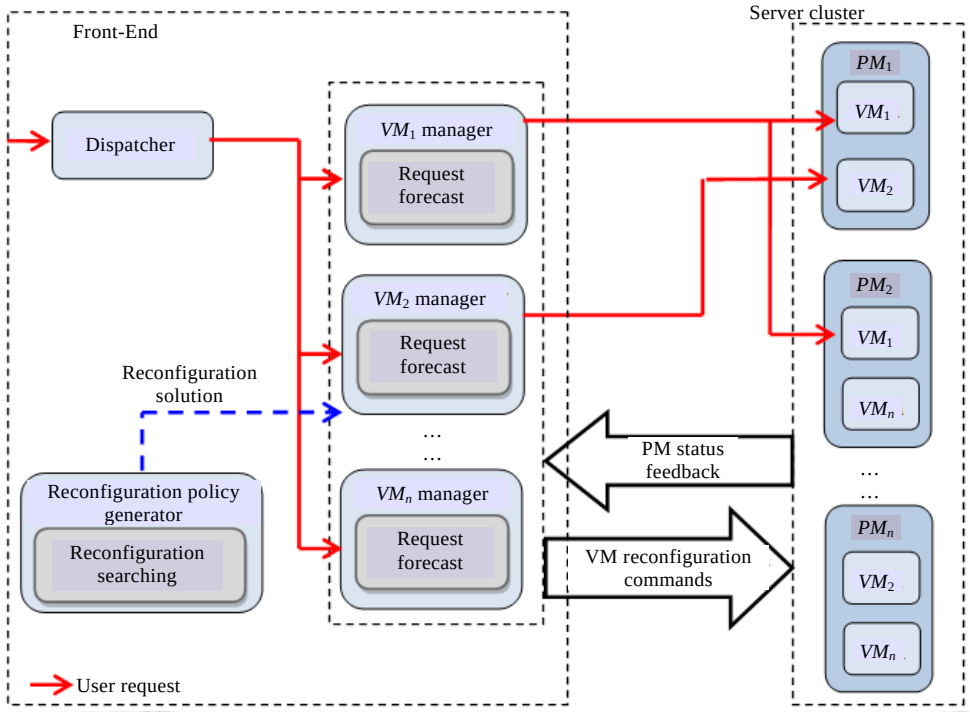


Fig.3 Self-Configuration architecture

图 3 自配置体系结构

在自配置框架中,请求预测模块和重配置搜索模块是两个核心子模块,预测的准确度和重配置搜索的速度及结果至关重要.本文采用布尔二次指数平滑法[16]实现请求预测模块,基于遗传算法[17]实现重配置搜索模块.

2 算法描述

由于部署在虚拟机中的 Web 应用包含多种类型的请求,每种类型的请求对资源的需求量不同,基于请求级的粒度[15]对虚拟机进行调度会过于复杂,因此我们采用应用级的粒度进行调度,定义节点的应用级响应能力.

定义 1(应用级响应能力). 当节点 i 仅部署虚拟机 j 时,每秒钟响应虚拟机 j 中各种类型请求的最大平均次数,其表达式为

$$\overline{req_{ij}}=\sum_{q\in Q_j}\alpha_q\times req_{iq} \tag{1}$$

其中,$Q_j=\{q_1,q_2,\ldots,q_n\}$表示请求类型的集合,$\alpha_q$为请求 q 所占比例,req_{iq}为节点 i 对请求 q 每秒钟的最大响应次数.

如前所述,重配置的目标是快速地搜索出最优配置以提高资源利用率,降低能耗.这一过程可以建模为多目标优化问题.为了简化分析,本文仅考虑节点的 CPU 资源.那么对每一次重配置,其形式化公式可以表示为

$$\begin{aligned}&\text{Min } F_{power}\\&\text{Max } \overline{U_{CPU}}\\&\text{s.t:}\\&\sum_{i=1}^{N} r_{ij} = load_j(t+\tau),\ \forall j \in M\\&\sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}} \leqslant U_{CPU\_upthreshold},\ \forall i \in N\end{aligned} \tag{2}$$

其中,r_{ij}表示节点 i 接收的对虚拟机 j 的请求,当 r_{ij}=0 时,表示请求数为 0,则节点 i 不启动虚拟机 j;$\sum_{i=1}^{N} r_{ij}$ 表示为集群接收的对虚拟机 j 的请求之和,其值等于预测值 $load_j(t+\tau)$;N 表示节点数;$\frac{r_{ij}}{req_{ij}}$ 表示虚拟机 j 对节点 i 的 CPU 需求;$\sum_{j=1}^{M} \frac{r_{ij}}{req_{ij}}$ 表示节点 i 中的虚拟机对其 CPU 的需求之和,其值应小于节点 i 的 CPU 资源的上限 $U_{CPU\_upthreshold}$;M 表示虚拟机的种类;$\sum_{j=1}^{M} r_{ij}$ 表示节点 i 接收的所有请求,当 $\sum_{j=1}^{M} r_{ij} = 0$ 时,表示节点 i 处于待机状态;F_{power} 和 $\overline{U_{CPU}}$ 分别表示集群能耗和 CPU 平均利用率,其设计细节将在第 2.2.2 节介绍.

2.1 请求预测

为使重配置结果能够持续满足用户请求对资源的需求,需对未来的用户请求进行预测.有多种预测方法能够满足本文的场景,如趋势外推、回归分析等,本文选用布尔二次指数平滑法[16].对虚拟机 j,预测公式为

$$\begin{cases} load_j(t+\tau) = a_t + b_t, \tau = 1,2,3,... \\ a_t = 2L_t^{(1)} - L_t^{(2)} \\ b_t = \dfrac{\alpha}{1-\alpha}(L_t^{(1)} - L_t^{(2)}) \\ L_t^{(1)} = \alpha load_j(t) + (1-\alpha)L_{t-1}^{(1)} \\ L_t^{(2)} = \alpha L_t^{(1)} + (1-\alpha)L_{t-1}^{(2)} \end{cases} \tag{3}$$

其中,$load_j(t)$为 t 时刻观察到的对虚拟机 j 的请求数,$load_j(t+\tau)$为 $t+\tau$时刻的预测值,$L_t^{(1)}$ 和 $L_t^{(2)}$ 分别为一次指数平滑值和二次指数平滑值,α为平滑参数.通过公式(3),可以在 t 时刻对 $t+\tau$时刻的请求进行预测,取值为

$$L_j(t+\tau)=load_j(t+\tau)+\varepsilon_j \tag{4}$$

其中,ε_j为预测误差.

2.2 重配置算法

由公式(2)可知,搜索最优配置的过程为 NP-难问题.为快速搜索出最优近似解,本文基于遗传算法设计搜索过程.依据遗传算法的思想,本文将每个配置方案看作一个染色体,根据其适应值的大小确定其优越性.适应值越高,生存率越大,作为父代繁殖的概率也越大.经过若干代的选择,筛选出最优配置方案.

2.2.1 染色体编码

每个染色体代表一个候选方案,表明虚拟机重配置后运行的位置和接收的请求数.染色体由多个不同的基因串构成,每个基因串对应一种虚拟机.每个基因串由若干个子串构成,每个子串对应一个节点.具体的二进制编码规则为:

1) 根据不同节点对虚拟机 j 的应用级响应能力确定子串的编码长度,例如,若节点 i 的响应能力为 200/s,则子串长度 l_{ij} 为 8 位.
2) 根据节点数量确定基因串的长度 $l_j = \sum_{i=1}^{N} l_{ij}$.
3) 根据虚拟机种类确定染色体基因串的数量.

以 3 种虚拟机、2 个节点为例给出编码说明:设节点对虚拟机的响应能力及对应的编码位数见表 1.这 3 种虚拟机构成 3 个基因串,每个基因串有两个子串.对虚拟机 1,节点 1 和节点 2 响应能力为 100/s 和 200/s,则对应

子串编码长度分别为 7 和 8,则虚拟机 1 对应的基因串长度为 15.相应地,虚拟机 2 和虚拟 3 对应的基因串包含两个子串,其长度分别为 17 和 18.

Table 1 PMs response ability to three kinds of VMs and corresponding coding length

表 1 节点对不同虚拟机资源请求的响应能力及对应编码长度

Node Num	Req1/s	Coding length	Req2/s	Coding length	Req3/s	Coding length
1	100	7	200	8	300	9
2	200	8	300	9	400	9

染色体编码如图 4 所示.

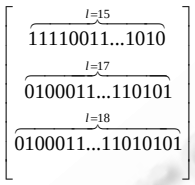


Fig.4 Chromosome coding

图 4 染色体编码

如此设计染色体的优势在于:(1) 当染色体对之间交叉、变异时,相同位置对应的基因串可同步隔离进行,这种方式可以在保持各应用之间独立性的前提下,大大加快染色体进化的速度;(2) 在配置资源时,充分考虑节点负载能力的限制.

2.2.2 适应值函数

本文中,适应值函数包括两个部分:耗能函数和惩罚函数.在一次请求分配中,我们追求集群能耗的最小值.对节点 i,能耗函数定义为

$$F_{power\_i}=\begin{cases}\tau\times\left(\sum_{j=1}^{M}\dfrac{r_{ij}}{req_{ij}}\times P_{Bi}+\left(1-\sum_{j=1}^{M}\dfrac{r_{ij}}{req_{ij}}\right)\times P_{Ii}\right), & \text{if } \exists r_{ij}\neq 0\\ \tau\times P_i^{off}, & \text{else}\end{cases} \tag{5}$$

其中,P_{Bi},P_{Ii} 和 P_i^{off} 分别为节点 i 工作、空闲和待机时的功率.在监控时间间隔 τ 内,节点 i 响应各虚拟机请求之后处于空闲状态,其能耗为 $\left(1-\sum_{j=1}^{M}\dfrac{r_{ij}}{req_{ij}}\right)P_{Ii}$.当节点 i 没有启动任何虚拟机时处于待机状态,此时能耗为 $\tau * P_i^{off}$.则集群的耗能函数定义为

$$F_{power}=\sum_{i=1}^{N}F_{power\_i} \tag{6}$$

为了尽快搜索出最优配置,我们采用惩罚函数来加快进化速度,惩罚函数定义为

$$F_{punish}=\sum_{j=1}^{M}F_{punish\_j}=\sum_{j=1}^{M}\alpha_j\times\frac{\tau\times\sum_{i=1}^{N}\dfrac{\left|L_j(t+\tau)-\sum_{i=1}^{N}r_{ij}\right|}{req_{ij}}\times P_{Bi}}{N} \tag{7}$$

其中,α_j 表示虚拟机 j 的惩罚悉数.集群中运行节点的平均 CPU 利用率定义为

$$\overline{U_{CPU}}=\frac{\sum_{i\in N}\sum_{j=1}^{M}\dfrac{r_{ij}}{req_{ij}}}{\left|\sum_{i\in N}i\right|} \tag{8}$$

由于适应度函数往往寻求最大适应值,因此将其定义为

$$F_{fitness}=\begin{cases}C_{\max}-F_{power}-F_{punish}, & \text{if } \overline{U_{CPU}}>U_{CPU\_downthreshold} \text{ and } \overline{U_{CPU}}<U_{CPU\_upthreshold}\\ C_{\text{mix}}, & \text{others}\end{cases} \tag{9}$$

其中,$U_{CPU\_upthreshold}$ 和 $U_{CPU\_downthreshold}$ 分别为 CPU 利用率的上下限,C_{mix} 和 $C_{\max}$ 为常量.其配置算法主要过程见算法 1,主要说明进化部分和适应值计算部分.

算法 1.

//*M*:虚拟机的种类; *N*:节点数量; generations:迭代次数; population:群体大小

//List⟨char[]⟩chromos:染色体基因串,包括 *M* 个子串.

//List⟨chromos⟩clist:群体个数,长度为 population

输入:workload[*M*]:前端接收到的对各类虚拟机的总请求数

输出:server_load[*j*][*i*]:集群中各节点分配到的各类虚拟机的请求数

```
1:  Initialize clist;
2:  for each generation in generations
3:    for each chromos in clist
4:      //根据 M 和 N 将染色体解码转化为十进制数值
5       for (i=1; 1<=chromos.sizes; i++)
6:        str=String.valueof(chromos.get(i));   //字符数组转化为字符串,chromos 对应虚拟机及其请求
7:        for (j=1; j<servers.length; j++)
8:          server_load[j][i]=str.substring(serverType.length);   //从字符串依次取出每个 server 分配虚拟机
            及其请求负载
9:        end for
10:     end for
11:     根据公式(9)计算计算适应值
12:   end for
13:   selection();   //根据适应值的大小选择优良的父代,结果保存在 clist
14:   crossover();   //在新个体中随机配对,在配对个体中随机选择交叉位,结果保存在 clist
15:   mutation();   //在新个体中随机选择变异点,结果保存在 clist
16: end for
```

3 实验与结果分析

数据中心资源按需动态配置的效果与请求预测算法、配置调整算法密切相关,如前所述,本文所提出自配置方法包括两个核心部分:预测算法和重配置算法.为进一步阐述所提方法的有效性,本节主要进行 4 类实验:

(1) 分析预测算法的准确度.

(2) 分析在节点数量和应用种类变化的条件下重配置算法的收敛性,并分析计算复杂度.

(3) 在不考虑请求预测的条件下与文献[12]对比重配置算法的扩展性.文献[12]提出的算法是面向大规模数据中心资源调度最具代表性的工作,可以很好地应用于虚拟化数据中心的场景,本文主要与该工作进行对比.

(4) 在考虑请求预测条件下与文献[12]对比整体方案的有效性.

3.1 实验环境

通常,数据中心由多个集群构成,重配置一般限制在单个集群内部,一个集群的典型规模在 50~300 节点之间,因此,本节实验模拟由 300 个节点构成的集群,其中,100 台 CPU 分别为 Intel(R) Core(TM)2 Duo 2.83GHz 的 Dell 主机、100 台 CPU 分别为 Intel(R) Core(TM)2 Duo 2.33GHz 的 Dell 主机和 100 台 CPU 为 AMD Athlon(TM)

64 X2 3600+ 1.9GHz 的 lenovo 主机.应用服务器采用 Tomcat6.0;操作系统为 Red Hat 2.6.24.3;虚拟机为 VMWare Station6.5;节点的耗电量见表 2,实验未考虑虚拟机对内存的要求.

Table 2 Node power consumption

表 2 节点耗电量

PM	P_{busy}(W)	P_{idle}(W)	P_{off}(W)
Dell (2.83GHz)	268	155	10
Dell (2.33GHz)	225	112	10
Lenovo (1.9GHz)	189	101	10

表 3 列出了遗传算法关键参数的经验值.这些参数的不同取值将对算法的效率和效果产生十分重要的影响.受篇幅所限,本文并不讨论如何选择最合适的参数值.

Table 3 Default values of key parameters

表 3 关键参数默认值

Parameter	Value
Population size	50
Selection method	Roulette wheele[24]
Crossover rate	0.9
Mutation rate	0.1

3.1.1 Web 应用类型

Web 应用类型采用 TPC-W 性能测试基准[18]中的 3 类典型应用:Browsing Mix(BM),Shopping Mix(SM)和 Ordering Mix(OM).在采用基于 Java 实现的 TPC-W 基准开源包部署之后,使用性能测试工具 Httperf[19]分别对 3 种应用进行压力测试,各节点对不同类型应用的应用级响应能力见表 4,监控时间间隔为 10m.

Table 4 Nodes application-level response capability

表 4 节点应用响应能力

	Req/s (BM)	Req/s (SM)	Req/s (OM)
Dell (2.83GHz)	235	197	130
Dell (2.33GHz)	190	163	91
Lenovo (1.9GHz)	132	89	57

3.1.2 用户请求分布

实验采用 3 种数据样本度量用户请求的变化:1) 1998 年 6 月 22 日 12:00AM~24:00PM 足球世界杯期间用户访问的请求记录[5];2) 1995 年 9 月 29 日 12:00AM~24:00PM 用户访问 ClarkNet Web 服务器的请求记录[20].为了产生足够的请求负载,两种请求数量都扩大了 10 倍;3) 模拟生成服从泊松分布的访问请求记录.3 种请求的分布如图 5 所示,每 10 分钟采样一次,共 72 采样点.

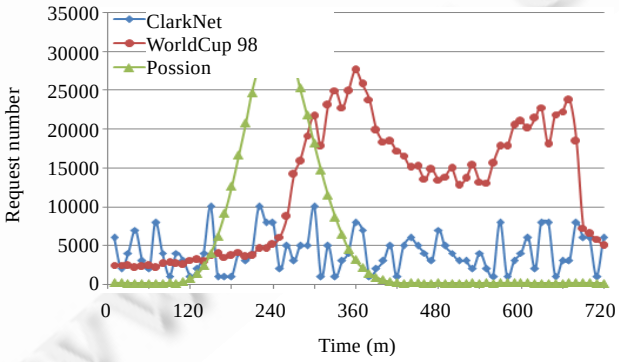


Fig.5 Changes of request distribution

图 5 请求分布变化

3.2 实验结果

3.2.1 预测算法准确度

以 World Cup98 请求分布为例,τ=10m,α=[0.1~0.9],则α=0.3,0.5,0.8 时预测分布与原请求分布之间的比较如图 6 所示.通过平均相对误差分析可知,α的取值对预测结果影响不大,如α=0.3 时平均相对误差为 0.078; α=0.5 时平均相对误差为 0.074;α=0.8 时平均相对误差为 0.1.由此可知,布尔二次预测算法能够较为准确地预测变化波动较大的请求,在以后的实验中取α=0.5.

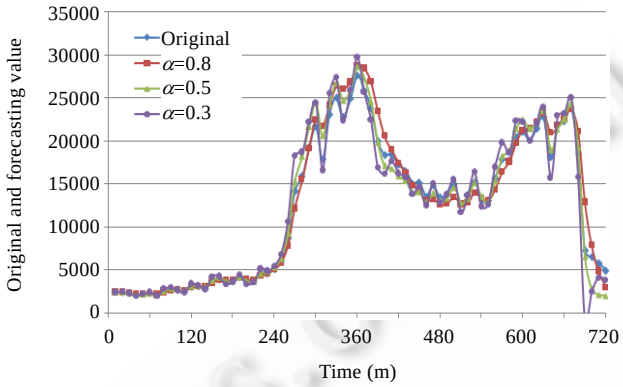


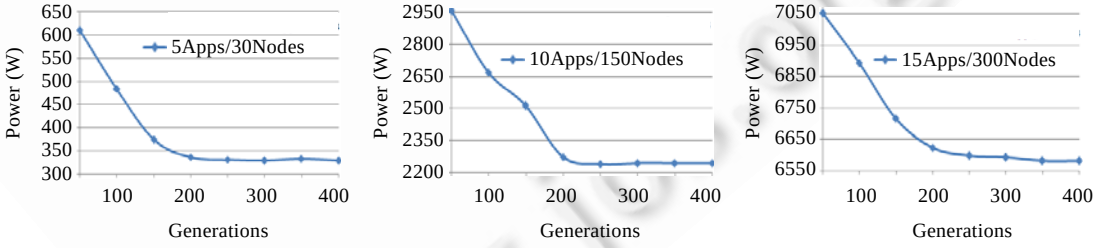
Fig.6 Real value and forecast value

图 6 请求真实值与预测值

3.2.2 重配置算法的收敛性和计算复杂度

重配置算法的收敛速度和时间开销受到节点数量和应用种类的影响.因此,实验的主要目的在于验证在各种节点数量与应用类型的组合下算法是否具有稳定的收敛性.实验条件:节点数取值范围 30~300,3 类节点各占 1/3;应用类型的取值范围 3 种~15 种,类型根据表 4 节点对 BM 类型的响应能力随机产生;所有应用的请求数量恒定,对资源的总需求量占集群总资源的 20%.

通过实验发现,重配置算法在各种节点数量与应用类型的组合下平均 250 次迭代即可达到较稳定的收敛.图 7(a)~图 7(c)分别显示了 5 种应用 50 台节点、10 种应用 150 节点和 15 种应用 300 台节点的条件下算法的收敛效果.



(a) 5 Applications and 30 nodes
(a) 5 种应用、30 个节点

(b) 10 Applications and 150 nodes
(b) 10 种应用、150 个节点

(c) 15 Applications and 300 nodes
(c) 15 种应用、300 个节点

Fig.7 Convergence effect of different applications and nodes

图 7 不同应用和节点数量条件下算法收敛效果

由图 5 可知,重配置算法的计算复杂度由节点数量 N、虚拟机种类 M、进化次数和群体规模决定.其中,初始群体规模设定为固定值(见表 3).由上述分析可知,当进化次数为 250 次算法可收敛,即进化次数亦可设定为固定值.因此,重配置算法的计算复杂度可近似地由节点数量和虚拟机种类决定,即 $O(MN)$.

图 8 和图 9 所示在各种节点数量与应用类型的组合下,算法迭代 250 次的时间开销.从图 8 中可以发现,当应用种类确定时,时间开销基本上随节点数的增加呈线性增长;通过相关分析后发现,两者的线性相关度约为 0.97.从图 9 可以发现,当节点数量确定时,时间开销基本上随节点数的增加呈线性增长;回归分析后发现,两者的线性相关度约为 0.99.上述数据可以说明,当虚拟机种类或节点数量固定时,重配置算法搜索的时间复杂度将降低为 $O(n)$.

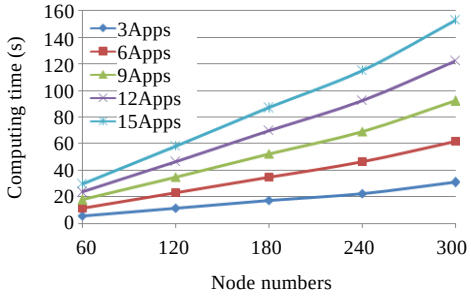


Fig.8 Computing time as nodes number changes

图 8 节点变化时的计算时间

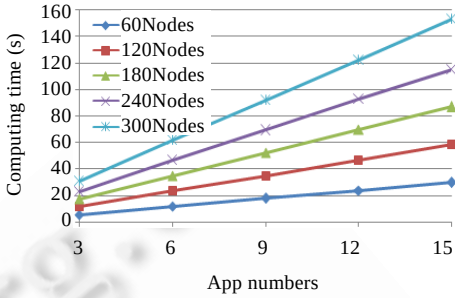


Fig.9 Computing time as application number changes

图 9 应用变化时的计算时间

3.2.3 算法扩展性对比

本节实验的目的在于不考虑预测算法的条件下,当集群中的节点数量扩大时,对比重配置算法与文献[12]的算法在调整效果方面的优劣.在图中本文的算法以“GABA”标识,文献[12]的算法以“TSSP07”标识.实验条件:节点变化范围 90~300,3 类节点各占 1/3;1 种应用,类型为表 4 中的 BM,应用对资源的总需求量占集群总资源的 60%;进化 250 代.

图 10~图 12 显示了随着节点数量的扩大,两种算法在节点的运行节点、CPU 利用率和能耗方面的对比.

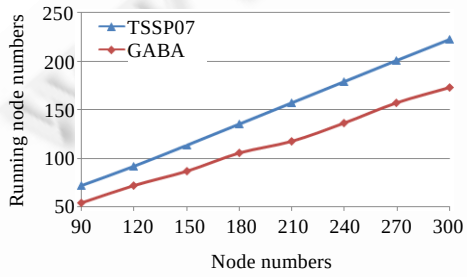


Fig.10 Comparison of running nodes

图 10 运行节点对比

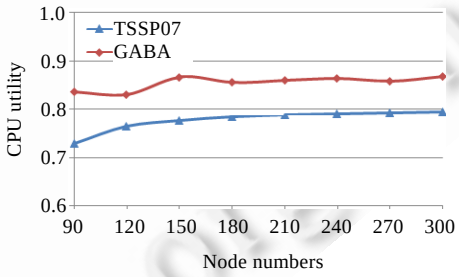


Fig.11 Comparison of CPU utilization

图 11 CPU 利用率对比

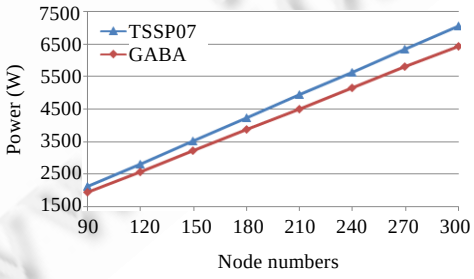


Fig.12 Comparison of power consumption

图 12 耗能对比

由实验条件可知,应用对资源的需求量随节点数目的增加呈线性增长趋势.从图 10 可以看出,GABA 与 TSSP07 计算出的运行节点数随着节点数目的增加亦呈线性增长趋势.这说明两种算法都具有高扩展性,但 GABA 的平均增长趋势仅为 TSSP07 的 3/4.从图 12 可知,随着运行节点数和应用对资源需求的线性增加,GABA 与 TSSP07 计算出的耗电量亦呈线性增加,但是 GABA 平均耗电量较 TSSP07 减少了 10%.从图 12 可知,虽然两种算法都可以使集群保持较高的 CPU 平均利用率,但 GABA 比 TSSP07 提高约 8%.

3.2.4 整体方案对比

如前所述,数据中心资源按需动态配置的效果与请求预测算法、配置调整算法密切相关.本节实验的目的是,在一定数量的节点构成的集群中,在虚拟机类型恒定的情况下(应用类型恒定),GABA 与 TSSP07 对比调整效果的优劣.实验条件:节点数为 300 个,3 类节点各 100 个;3 类应用,类型见表 4,请求分布如图 5 所示.

图 13~图 15 分别显示了 GABA 与 TSSP07 在 12 小时期间节点运行数量、CPU 利用率和能耗方面的对比.如图 13 所示,TSSP07 的调整策略为当前配置能够满足应用的请求时就不再调整集群.所以从整体看,TSSP07 较 GABA 开启了许多不必要的节点.尤其在 300m 之后,由于请求峰值致使集群中的所有节点全部开启,在之后的时间内 TSSP07 不再调整,所以使不必要开启的节点大大增加,这使得 CPU 利用率下降,能耗大幅增加,如图 14 和图 15 所示.这也反映出当前多数数据中心按照请求峰值分配资源的不合理性.相反,GABA 根据不断变化的需求对集群资源进行重配置,使其可以以最小的资源满足需求.在 12 小时期间采用 GABA 平均运行的节点数为 133,而采用 TSSP07 平均运行的节点数为 255.如图 14 所示,因为整个时间段 TSSP07 开启了众多不必要的节点,致使集群中运行节点的 CPU 平均利用率只有 53.4%;而采用 GABA,CPU 平均利用率提高了 35%.图 15 描述了整个时间段的能耗,TSSP07 共消耗了 536.2KW;而 GABA 消耗了 409.3KW,较 TSSP07 节省了 25%.

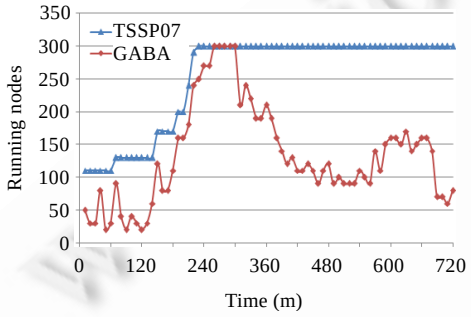


Fig.13 Comparison of running nodes

图 13 运行节点对比

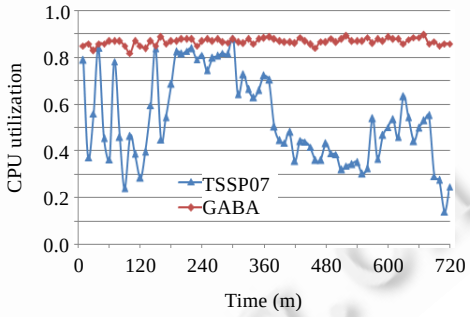


Fig.14 Comparison of CPU utilization

图 14 CPU 利用率对比

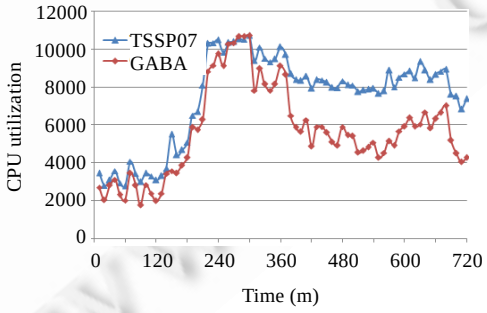


Fig.15 Comparison of power consumption

图 15 能耗对比

4 相关工作

目前,有很多研究试图解决数据中心的资源动态配置问题[3,4,6−15].Walsh 等人[6]采用服务级和资源级两层效用函数解决数据中心异构节点之间的资源自配置问题,为每种应用分配一个应用管理器,各个应用管理器根据服务级效用函数计算应用对资源的需求,并将结果提交给资源仲裁器,资源仲裁器根据资源级效用函数计算各个应用需要的节点数量.Das 等人[7]在此工作的基础上,利用 Agent 技术实现了一个数据中心能耗管理的系统原型;Tesauro 等人[8]将数据中心的资源分配问题建模为组合马尔可夫过程[21],并基于 Q 值强化学习算法[10]提出了一种资源在线分配模型.然而,这些研究都不允许应用之间共享节点资源,即假设一个节点只能部署一种应用.这种假设不适合虚拟化的数据中心.

Bobroff 等人[4]提出一种基于虚拟机动态迁移的方法来关闭不必要启动的节点.该方法首先采用线性时序预测方法[22]来预测虚拟机对资源的需求,并根据需求对虚拟机降序排列,然后采用首次满足(first-fit)背包算法将虚拟机部署在合适的节点上.然而,该研究仅仅考虑为每个应用分配一个虚拟机的场景,并不适用数据中心中一种虚拟机有多个副本的场景.Kusic 等人[9]基于有限控制预测技术[11]提出一个动态资源分配框架,通过两层控制体系来确定虚拟机需要启动的副本数、副本所在节点的位置和位于同一节点上的虚拟机分配的资源量.虽然该方法通过关闭不必要开启的节点提高了资源利用率,但是该方法的计算复杂度为指数级.当节点为 5 个时,决策时间小于 10s;而节点为 15 个时,决策时间接近 30m.显然,该方法不适用于数据中心.

Karve 等人[13]提出了一种针对大规模数据中心的应用动态放置方法,可以根据应用不断变化的资源需求动态确定应用所在的节点位置.他们将该问题建模为多目标优化问题,采用 3 种算法(剩余配置算法、增量配置算法和重平衡配置算法)搜索近似最优配置.剩余配置算法按照升序排列应用资源需求列表和节点服务能力列表;增量配置算法将变化的需求重分配至各个节点;重平衡配置算法负责平衡各节点之间的负载.Tang 等人[12]采用隔离的方式进一步降低了文献[13]中算法的计算复杂度,即当考虑是否改变某一节点上的应用请求时,假设其他节点的状态不改变.这些方法的不足在于:第一,采用的启发式算法虽然简化了搜索算法的计算复杂度,降低了搜索时间,但其片面追求了速度而忽视了重配置效果;第二,算法执行的时机存在缺陷,假设当前配置能够满足新的需求时不再调整配置的结构;第三,没有考虑关闭或休眠不必启动的节点以减少能耗.与之相比,本文提出的方法在保证搜索速度的同时又兼顾重配置效果,同时亦考虑关闭空闲节点来减少能耗.

另外,上述各工作仅根据当前时刻点的请求计算重配置方案,忽视节点调整的时间开销,致使调整结果滞后于请求变化.文献[4,9]虽对请求作了预测,但其采用了简单的线性预测方法,不适合需求变化波动较大的场景.

我们首先在文献[23]中提出了基于遗传算法的资源重配置方案.遗传算法是一种基于生物自然选择与遗传机理的随机搜索算法[24],广泛的应用于最优化问题中[17].与传统优化算法相比,遗传算法具有以下特点:1) 对参数集进行编码处理,使遗传算法可以直接操作诸如集合、序列、矩阵等数据对象;2) 并行化搜索,同时对搜索空间中多个可行解评估;3) 利用适应值信息,无需搜索空间的知识和其他辅助信息;4) 利用概率转移规则指导搜索方向,能够有效进行概率意义上的全局搜索.遗传算法已经较好地应用于数据镜像[25]、动态网络拓扑重构[26]、网络编码优化[27]等领域.本文针对不同规模的实验环境,就算法的收敛性、可扩展性等进行了进一步探讨.

5 结束语

当前能源开支成为制约数据中心发展的关键因素,如何有效地提高资源利用率成为数据中心关注的重要问题.本文面向大规模虚拟化集群环境,提出一种集群资源按需动态配置方法.该方法能够根据不断变化的需求和环境条件,在线、高效地调整集群中节点运行的数量及运行节点上部署的虚拟机种类.该方法基于布尔二次指数平滑法预测用户请求,能够有效避免重配置之后的结构滞后于请求;基于遗传算法并行搜索配置空间,能够快速搜索出最优配置.实验结果表明,算法能够根据变化的环境快速做出调整,有效提高集群利用率,降低能耗.

需要说明的是:1) 本文着重研究的是单个集群内部的资源按需配置问题.当考虑资源跨集群动态配置时,如何有效解决多个前端环境下的分布式决策冲突,将是研究的重点;2) 本文假设各 Web 应用之间相互独立,而

现实集群中的应用亦大量存在着彼此关联的情况,这将导致集群中的虚拟机存在相互依赖的关系.这种场景下,调度将会变得更加复杂,也是我们下一步关注的重点.

References:

[1] Gartner Inc. Gartner Says 50 Percent of Data Centers Will Have Insufficient Power and Cooling Capacity by 2008. Press Release, 2008.

[2] McKinsey & Co. Report. http://uptimeinstitute.org/content/view/168/57

[3] Padala P, Shin KG, Zhu XY, Uysal M, Wang ZK, Singhal S, Merchant A, Salem K. Adaptive control of virtualized resources in utility computing environments. ACM SIGOPS Operating Systems Review, 2007,41(3):289−302. [doi: 10.1145/1272996.1273026]

[4] Bobroff N, Kochut A, Beaty K. Dynamic placement of virtual machines for managing SLA violations. In: Proc. of the 10th IFIP/IEEE Int'l Symp. on Integrated Network Management. 2007. 119−128. [doi: 10.1109/INM.2007.374776]

[5] Arlitt M, Jin T. Workload characterization of the 1998 World Cup Web site. Technical Report, HPL-99-35R1, Hewlett-Packard Labs., 1999.

[6] Walsh WE, Tesauro G, Kephart JO, Das R. Utility functions in autonomic systems. In: Proc. of the 4th Int'l Conf. on Autonomic Computing. 2004. 70−77. [doi: 10.1109/ICAC.2004.1301349]

[7] Das R, Kephart JO, Lefurgy C, Tesauro G, Levine DW, Chan H. Autonomic multi-agent management of power and performance in data centers. In: Proc. of the 7th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems. 2008. 107−114.

[8] Tesauro G. Online resource allocation using decompositional reinforcement learning. In: Proc. of the 20th National Conf. on Artificial Intelligence. 2005. 886−891.

[9] Kusic D, Kephart JO, Hanson JE, Kandasamy N, Jiang GF. Power and performance management of virtualized computing environments via lookahead control. Journal of Cluster Computing, 2009,12(1):1−15. [doi: 10.1007/s10586-008-0070-y]

[10] Russell S, Zimdars AL. Q-Decomposition for reinforcement learning agents. In: Proc. of the 20th Int'l Conf. on Machine Learning. 2003. 656−661.

[11] Abdelwahed S, Kandasamy N, Neema S. Online control for self-management in computing systems. In: Proc. of the Real-Time and Embedded Technology and Applications Symp. 2004. 368−375. [doi: 10.1109/RTTAS.2004.1317283]

[12] Tang CQ, Steinder M, Spreitzer M, Pacifici G. A scalable application placement controller for enterprise data centers. In: Proc. of the 16th Int'l Conf. on World Wide Web. 2007. 331−340. [doi: 10.1145/1242572.1242618]

[13] Karve A, Kimbrel T, Pacifici G, Spreitzer M, Steinder M, Sviridenko M, Tantawi A. Dynamic placement for clustered Web applications. In: Proc. of the 15th Int'l Conf. on World Wide Web. 2006. 593−604. [doi: 10.1145/1135777.1135865]

[14] Weng CL, Li ML, Wang ZG, Lu XD. Automatic performance tuning for the virtualized cluster system. In: Proc. of the 29th IEEE Int'l Conf. on Distributed Computing Systems. 2009. 183−190. [doi: 10.1109/ICDCS.2009.45]

[15] Heath T, Diniz B, Carrera EV, Jr. Meira W, Bianchini R. Energy conservation in heterogeneous server clusters. In: Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. 2005. 186−195. [doi: 10.1145/1065944.1065969]

[16] Brown RG, Meyer RF. The fundamental theorem of exponential smoothing. Journal of Operations Research, 1961,9(5):673−685.

[17] Back T, Hammel U, Schwefel HP. Evolutionary computation: Comments on the history and current state. IEEE Trans. on Evolutionary Computation, IEEE, 1997,1(1):3−17. [doi: 10.1109/4235.585888]

[18] Menasc DA. TPC-W: A benchmark for e-com merce. IEEE Internet Computing, 2002,6(3):83−87. [doi: 10.1109/MIC.2002.1003136]

[19] Mosberger D, Jin T. Httperf—A tool for measuring Web server performance. ACM SIGMETRICS Performance Evaluation Review, 1998,26(3):31−37. [doi: 10.1145/306225.306235]

[20] Arlitt MF, Williamson CL. Web server workload characterization: The search for invariants. In: Proc. of the ACM SIGMETRICS Conf. on the Measurement and Modeling of Computer Systems. 1996. 23−26. [doi: 10.1145/233013.233034]

[21] Meuleau N, Hauskrecht M, Kim KE, Peshkin L, Kaelbing LP, Dean T, Boutilier C. Solving very large weakly coupled Markov decision processes. In: Proc. of the 13th National Conf. on Artificial Intelligence. 1998. 165−172.

[22] Wei WW. Time Series Analysis. Addison-Wesley, 1990. 23−26.

[23] Mi H, Wang HM, Yin G, Zhou YF, Shi DX, Yuan L. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In: Proc. of the 2010 IEEE Int'l Conf. on Services Computing. 2010. 514−521. [doi: 10.1109/SCC.2010.69]

[24] Fogel DB. An introduction to simulated evolutionary optimization. IEEE Trans. on Neural Networks, 1994,5(1):3−14. [doi: 10.1109/72.265956]

[25] Ramirez AJ, Knoester DB, Cheng BHC, McKinley PK. Applying genetic algorithms to decision making in autonomic computing systems. In: Proc. of the 6th Int'l Conf. on Autonomic Computing. 2009. 97−106. [doi: 10.1145/1555228.1555258]

[26] Montana D, Hussain T. Adaptive reconfiguration of data networks using genetic algorithms. Journal of Applied Soft Computing, 2004,4(4):433−444. [doi: 10.1016/j.asoc.2004.02.002]

[27] Deng L, Zhao J, Wang X. Genetic algorithm solution of network coding optimization. Journal of Software, 2009,20(8):2269−2279 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3370.htm [doi: 10.3724/SP.J.1001.2009.03370]

附中文参考文献:

[27] 邓亮,赵进,王新.基于遗传算法的网络编码优化.软件学报,2009,20(8):2269−2279. http://www.jos.org.cn/1000-9825/3370.htm [doi: 10.3724/SP.J.1001.2009.03370]

米海波(1982－),男,河南新乡人,博士生,CCF 会员,主要研究领域为分布计算,云计算,性能分析.

史殿习(1966－),男,博士,副研究员,CCF 会员,主要研究领域为分布计算,移动云计算,软件工程.

王怀民(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布计算,云计算,网络安全.

周扬帆(1977－),男,博士,主要研究领域为传感器网络,云计算,分布计算.

尹刚(1975－),男,博士,讲师,CCF 会员,主要研究领域为数据挖掘,分布计算.

袁霖(1981－),男,讲师,CCF 会员,主要研究领域为数据挖掘,分布计算,软件可信评估.