文章编号:1007－6735(2020)04－0404－07　DOI: 10.13255/j.cnki.jusst.20190722012

面向云数据中心资源均衡分配需求的聚类调度算法研究

徐雨婷，　陈世平
（上海理工大学 光电信息与计算机工程学院，上海 200093）

摘要：针对云数据中心资源利用率较低、能源消耗较高的问题，提出了基于资源需求差异的资源均衡调度策略。在包簇框架模型基础上，利用与资源需求相关的距离度量因子，将资源需求差异大的包通过改进的 *k*-means 算法进行聚类；利用资源之间的相关性作为包与簇之间的距离，在资源分配的过程中使包能够集中映射到簇中，从而减少簇的使用个数。实验结果表明，在包簇框架的概念下，基于资源需求差异的改进后的 *k*-means 聚类算法能够优化包聚类步骤，资源调度算法能够提高云数据中心各类资源利用率、降低资源分配过程中产生的能耗，具有有效性和可扩展性。

关键词：数据中心；包簇框架；*k*-means 算法；资源分配
中图分类号：TP 393　　**文献标志码：**A

Clustering scheduling algorithm for resource allocation requirements of cloud data centers

XU Yuting,　CHEN Shiping
(*School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China*)

Abstract: Aiming at the problem of low resource utilization and high energy consumption in cloud data center, a resource balancing scheduling strategy based on the resource demand difference was proposed. Based on a package-cluster framework model, the packages with large differences in resource requirements were clustered an improved *k*-means algorithm using the distance metrics related to resource requirements. The resources were used as the distance between packages and clusters. In the process of resource allocation, the package was mapped into clusters in a centralized manner, thereby the number of clusters used could be reduced. The experimental results show that under the concept of package-cluster framework, the improved *k*-means clustering algorithm based on the difference of resource requirements can optimize the packet clustering step. The resource scheduling algorithm presented can improve the utilization of various resources and reduce the energy consumption in the cloud data center. The algorithm is of effectiveness and scalability.

收稿日期: 2019−07−22
基金项目: 国家自然科学基金资助项目（61472256，61170277）；上海市一流学科建设项目（S1201YLXK）
第一作者: 徐雨婷(1993−)，女，硕士研究生．研究方向：计算机网络、计算机系统结构．E-mail：1260764947@qq.com
通信作者: 陈世平(1964−)，男，教授．研究方向：云计算、计算机网络通信、数据库与知识库．E-mail：chensp@usst.edu.cn

Keywords: *data center*; *package-cluster framework*; *k-means algorithm*; *resource allocation*

由于全球经济的发展，云计算已经成为社会的一种重要计算模式[1]，并在发展过程中形成了性能佳、费用低、可靠性强等一些独特的特点。随着云计算数据中心数据量的快速增长，人们开始研究数据中心在进行资源分配的过程中如何提高系统资源利用率的方法[2-3]。

已有许多学者针对虚拟机−服务器的资源映射问题进行了研究。Selim 等[4]利用 CPU 利用率差异概念，通过计算出所有物理服务器的 CPU 利用率的最小方差，选择最合适的主机分配虚拟机，但该算法仅考虑了 CPU 利用率，未考虑其他资源的使用情况。Combarro 等[5]针对三层数据中心提出了一种基于功率和网络感知调度模型，可通过资源整合和负载均衡调整的方式来实现降低能耗的目标，但未考虑资源利用率的优化。房丙午等[6]通过建立虚拟机放置模型，基于两个阶段的启发式算法来求解虚拟机放置模型，该启发式算法仅针对资源分配过程中的能耗问题进行研究，未考虑资源利用率的问题。

本文主要在包簇框架[7]基础上，研究数据中心资源调度策略。首先，基于包资源需求的分布密度，将资源需求（CPU 需求以及内存需求）具有差异性的包通过改进的 *k*-means 算法进行聚类；其次，通过皮尔逊相关系数得到资源的相关性，定义包与簇之间的距离，将包映射到最近的簇；最后，通过 Hadoop 框架和实验工具 CloudSim 分别进行实验，验证包聚类算法以及资源调度算法的有效性。

1　传统的 *k*-means 算法

传统的 *k*-means 算法具有良好的可扩展性，*k*-means 算法的思想主要是将聚类的中心点（质心）移动到聚类中包含其他样本点的平均位置，然后重新划分类。*k* 值是通过算法计算出来的参数，表示类的数量。*k*-means 的参数包括类的质心位置及其内部观测值的位置。传统的 *k*-means 算法过程如下[8]：

步骤 1　输入样本点集合，随机确定 *k* 个中心点；

步骤 2　计算样本集合中每个点到中心点之间的距离，将集合按照最小距离原则分配到最邻近聚类；

步骤 3　计算类中样本均值，更新每个类中心点的位置；

步骤 4　重复步骤 2 和步骤 3，直到所有中心不再更新或到达设定条件停止，输出 *k* 个类划分和最终的聚类中心。

2　包聚类算法

2.1　改进的 *k*-means 算法初始值选定

为了解决传统的 *k*-means 算法由于随机确定初始值对聚类结果造成影响的问题，需首先采用 canopy 算法确定 *k*-means 算法的初始值 *k* 及其初始中心[9]，通过 canopy 算法对数据的预处理，可提高 *k*-means 算法的处理速度，同时提高聚类准确性。

为了具体地实施包的聚类任务，采用基于 canopy 算法的 *k*-means 算法来实现包的聚类，将包的分布密度作为参数代入到 canopy 算法中，并定义与包的 CPU 需求、内存需求的相关距离度量因子，实现聚类过程。

将通过聚类形成的包简称为聚类包。现定义以下几种参数：

a. 集合中 2 个需求差异最大的包之间的距离 L。现假设集合中有点 $i:(x_{i,1},x_{i,2})$，$j:(x_{j,1},x_{j,2})$，其中，$x_{i,1}$和$x_{j,1}$表示包的 CPU 需求，$x_{i,2}$和$x_{j,2}$表示包的内存需求，若在某时刻这 2 个包的需求差异最大，则将它们之间的距离定义为集合中包之间的最大距离。

$$L=\sqrt{(x_{i,1}-x_{j,1})^2+(x_{i,2}-x_{j,2})^2} \tag{1}$$

b. 距离因子 $d(x_i,x_j)$。$d(x_i,x_j)$表示聚类的距离度量因子，x_i, x_j分别代表包的 CPU 需求和内存需求。

$$d(x_i,x_j)=\begin{cases}\left[\dfrac{x_{i,1}}{L}-\left(1-\dfrac{x_{j,1}}{L}\right)\right]^2+\left[\dfrac{x_{i,2}}{L}-\left(1-\dfrac{x_{j,2}}{L}\right)\right]^2, L>0\\ 0, L=0\end{cases} \tag{2}$$

当$L>0$时，可由式（2）计算点i与点j之间的距离因子。以 CPU 需求差异为例，若点i与点j的 CPU 需求 $x_{i,1}$与 $x_{j,1}$之间的差异越大，则 $x_{i,1}/L$与$(1-x_{j,1}/L)$的值越接近，则满足聚类原则中距离相近的点聚为一类的条件；当$L=0$时，表示集合中任意 2 个包之间的资源需求均相等，包之间不存在资源需求差异，此时$d(x_i,x_j)$=0。因此，若包之

间的CPU需求、内存需求差异越大，则2点之间的距离因子越近，最终需求差异大的2个包将处于1个聚类中。

c. 距离因子均值$D_{\rm is}$。$D_{\rm is}$为集合中所有包的距离因子的均值。n为集合中所有包样本点的数目。

$$D_{\rm is}=\frac{2\sum_{i=1}^{n}\sum_{j=1}^{n}d(x_i,x_j)}{(n-1)} \tag{3}$$

d. 包样本点分布密度$f(i)$。$f(i)$定义为集合中与包样本点i之间的距离因子小于$D_{\rm is}$的包样本点数目，表示点i的分布密度，所有满足条件的包样本点构成一个聚类。

$$f(i)=\sum_{i=1}^{n}\sum_{j=1}^{n}g(x_i,x_j) \tag{4}$$

$$g(x_i,x_j)=\begin{cases}1, & d(x_i,x_j)<D_{\rm is}\\ 0, & d(x_i,x_j)\geqslant D_{\rm is}\end{cases} \tag{5}$$

e. 聚类包中的距离因子均值$\theta(i)$。$\theta(i)$表示已经形成的聚类包的距离因子的平均值。

$$\theta(i)=\frac{2\sum_{i=1}^{f(i)}\sum_{j=1}^{f(i)}d(x_i,x_j)}{f(i)(f(i)-1)} \tag{6}$$

f. 2个不同分布密度的聚类包的距离因子$h(i)$。$h(i)$表示聚类包中心点i与其他聚类包中心点之间的距离因子。现存在2种情形：

(a)若存在聚类包中心点$j_1,j_2,\cdots,j_R$，R为聚类包的总个数，它们的分布密度均大于$f(i)$，分别计算点i与它们之间的距离因子，若点i与点$j_r(r\in R)$之间的距离因子$d(i,j_r)$最小，则$h(i)=d(i,j_r)$。

(b)若点i的分布密度$f(i)$大于任意一个聚类中心点的分布密度，分别计算点i与其他聚类中心点之间的距离，若点i与点b之间的距离因子$d(i,b)$最大，则$h(i)=d(i,b)$。

g. 聚类包的最大分布密度乘积μ。由上述分析可知，$f(i)$值越大，则i的周围的点就越多；$\theta(i)$值越小，聚类包中的点就越紧密；$h(i)$值越大，则2个聚类包的差异性就越大。由聚类的基本原则定义最大分布密度乘积μ，μ为聚类包中的距离因子均值、聚类包内的分布密度、聚类以后的包之间的距离的倒数这三者的乘积。

$$\mu=\frac{f(i)^2[f(i)-1]}{2\sum_{i=1}^{f(i)}\sum_{j=1}^{f(i)}d(x_i,x_j)}h(i) \tag{7}$$

2.2 基于资源需求差异的k-means算法

将基于资源需求差异的k-means算法流程分为两部分进行描述，第一部分为基于canopy算法确定k-means算法的k值和中心，第二部分为k-means的算法流程。

首先介绍第一部分的算法流程。

步骤1 定义由包组成的数据集D，计算集合中包的距离因子均值$D_{\rm is}$和包样本点分布密度$f(i)$。

步骤2 定义数据集A。步骤1计算出的具有最大分布密度值的包a_1被选择成为第一个聚类的中心点，将点a_1加入到数据集A中，将满足与初始类中心点a_1之间的距离因子小于$D_{\rm is}$条件的所有包样本点从D中删除。

步骤3 计算由步骤2确定的D中剩余包样本点的$\theta(i)$，以及$h(i)$值，然后根据式(7)计算点i与a_1之间的μ值，选择具有最大μ值所对应的点i，作为第二个聚类中心a_2，并将a_2放入集合A中。同样，将满足与a_2之间距离因子小于$D_{\rm is}$的包样本点从D中删除。

步骤4 计算D中剩余的样本点i与a_1和a_2的之间的μ值，分别记作μ_1和μ_2，选取具有最大分布密度乘积$(\mu_1\mu_2)$的点i，将点i作为第三个聚类的中心a_3，并将a_3放入集合A中。同样，将满足与a_3之间距离因子小于$D_{\rm is}$的包样本点从D中删除。

步骤5 同理，若包样本点j与聚类中心a_1，$a_2,\ldots,a_{k-1}$之间具有最大分布密度乘积$(\mu_1\mu_2\cdots\mu_{k-1})$，则包$j$将被设置成第$k$个聚类中心。

步骤6 循环步骤5，最终得到k值，以及初始类中心$a_i,i\in k$。

由上述步骤最终确定的k值以及初始中心a_i，代入到k-means算法中，执行k-means算法流程：

步骤1 计算集合中每个包样本点i到a_i的距离因子。

步骤2 将与a_i之间的距离因子最小的样本点分配到a_i所在的类中。

步骤3 计算每个类中的平均值，并更新聚类中心。

步骤4 重复执行步骤2和步骤3，直到所有点聚类完成。

2.3 示例说明

假设在t时刻，有8个待分配的包，如表1所示。假设包需求为(1, 1)的含义是对CPU的需求为1核，对内存的需求为1 MB。

表 1　包需求参数(1)

Tab.1　Package demand parameters(1)

包编号	包需求
1	(1, 1)
2	(1, 4)
3	(4, 1)
4	(4, 2)
5	(2, 1)
6	(3, 2)
7	(1, 2)
8	(4, 5)

a. 定义集合 D，并将表 1 中所有包需求参数放入集合 D 中。首先根据式(2)分别计算 8 个包的距离因子，根据式(3)计算出距离因子均值D_{is}，再根据式(4)计算包样本点分布密度$f(i)$。

b. 定义集合 A，将分布密度$f(i)$最大的包设为第一个聚类中心点，首先选择$f(i)$最大为 4 的包样本点，即编号为 7 的包样本点，作为第一个聚类中心，将其放入集合 A 中，并将满足与编号为 7 包样本点之间的距离因子小于D_{is}的样本点从 D 中删除，删除包编号为 3，4，6，8 的样本点。

c. 通过 b 的计算，D 中剩下包编号为 1，2，5 的点，再分别根据式(6)计算其$\theta(i)$，以及它们各自的$h(i)$，根据式(7)分别计算它们与 b 确定的编号为 7 的包样本点之间的μ，得出编号为 1 的包样本点具有最大μ，作为第二个聚类中心，并将其放入集合 A 中，并将满足与编号为 1 包样本点之间的距离因子小于D_{is}的样本点从 D 中删除，删除包编号为 2，5 的样本点，此时集合 D 为空，基于分布密度的 canopy 算法结束。

d. 执行 k-means 算法聚类流程。根据上述步骤得到 k 值为 2，初始类中心点为编号 1 和 7 的包。继续计算表 1 中其他样本点与它们之间的距离因子，将距离因子与中心点最小的样本点分配到它们所在的类中，计算平均值，更新类中心，根据 k-means 算法步骤，一直循环，直至聚类完成。

本示例形成了 2 个聚类包。第一个聚类包括编号 3，4，6，7，8 的包，其中，编号 3，4，6 为对 CPU 需求较高的包样本点，编号 7，8 为对内存需求较高的包样本点；第二个聚类包包括编号为 1，2，5 的包，其中，编号 5 为对 CPU 需求较高的包样本点，编号 2 为对内存需求较高的包样本点。形成的 2 个聚类包均为需求差异较大的包，由此可知，基于 canopy 算法的 k-means 算法可将资源需求差异较大的包聚类，从而提高各资源的利用率。最终得到的聚类结果如图 1 所示。

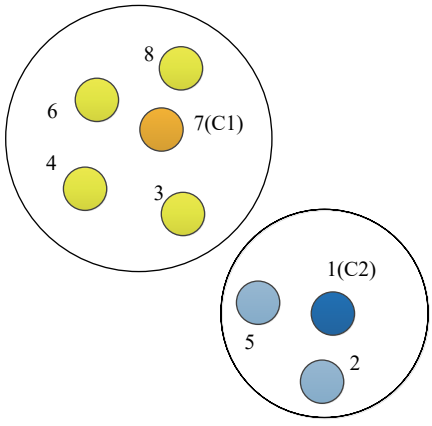

图 1　聚类以后的包

Fig. 1　Package after clustering

C1，C2 分别为第一个、第二个聚类包的聚类中心点。

3　包簇资源调度算法

3.1　包簇资源利用率及能耗定义

经过前面的聚类算法后，系统中共有 k 个待分配的聚类包，定义包集合为$\{x_1, x_2, \cdots, x_k\}$，定义簇集合为$\{y_1, y_2, \cdots, y_n\}$。现假设 t 时刻，包$v\ (1 \leqslant v \leqslant k)$的资源需求总量为$R_v[t]$，簇$p\ (1 \leqslant p \leqslant n)$可提供资源使用总量为$S_p[t]$，定义簇$p$中已使用资源总量为$W_p[t]$，规定每个簇的资源只能分配给一个包，即包中使用的资源不能超过簇中拥有的总资源，资源利用率

$$U = R_v[t]W_p[t]/S_p[t] \tag{8}$$

式中，$0 \leqslant U \leqslant 1$。

在资源分配过程中，除了需要考虑资源利用率以外，还需考虑资源分配过程中产生的能源消耗，可建立资源利用率与能耗之间的关系式[10]，式(9)为混合任务负载环境下的能耗模型。

$$E_p = \alpha_0 + \sum_{j=1}^{m} \alpha_j F_j\left(y_p\right) \tag{9}$$

式中：E_p为能耗；$F_j\left(y_p\right)$为E_p的核函数，由指数函数表示；y_p为各类资源(CPU、内存)利用率；α_0，α_j为核函数的参数，由模型训练可得；m为用于训练的数据集中的数据总数目。

3.2 包簇之间的距离计算

簇与聚类包之间的距离由资源之间的相关性进行定义，采用Pearson(皮尔逊)[11]相关系数描述簇中已使用的资源总量与包的需求总量之间的相关性，分别将$R_v[t]$，$S_p[t]$，$W_p[t]$代入相关系数，由于相关系数$\rho_{R_v(t),W_p(t)}$的因子取值为[0,1]，所以，将$R_v[t]$与$W_p[t]$进行归一化，再代入式(10)，可得

$$\rho_{R_v(t),W_p(t)}=\frac{\sum_t\left(\frac{R_v[t]}{S_p[t]}-\overline{\frac{R_v[t]}{S_p[t]}}\right)\left(\frac{W_p[t]}{S_p[t]}-\overline{\frac{W_p[t]}{S_p[t]}}\right)}{\sqrt{\sum_t\left(\frac{R_v[t]}{S_p[t]}-\overline{\frac{R_v[t]}{S_p[t]}}\right)}\sqrt{\sum_t\left(\frac{W_p[t]}{S_p[t]}-\overline{\frac{W_p[t]}{S_p[t]}}\right)}}$$

$$1\leqslant v\leqslant k,\ 1\leqslant p\leqslant n \tag{10}$$

由于相关系数的区间为[−1,1]，因此，定义簇与聚类之后的包之间的距离

$$L_{R_v(t),W_p(t)}=[\rho_{R_v(t),W_p(t)}+1]/2 \tag{11}$$

式(11)表示包簇之间的距离，区间为[0,1]，当由式(11)计算得出的$L_{R_v(t),W_p(t)}$值处于[0,1/2)时，表示$R_v[t]$与$W_p[t]$之间具有负相关性，具有负相关性的虚拟机与物理节点之间一般具有更多的资源互补[12-13]。因此，在进行包簇资源分配过程时，首先将包映射到与其距离的区间在[0,1/2)之间的簇，且在这区间中距离值越小，代表包与簇之间的资源互补性越大，最终可在资源分配时使得包能够集中映射到簇，从而减少簇的使用个数，提高资源利用率，降低分配过程中产生的能耗。

3.3 包簇资源调度算法的流程

基于改进的*k*-means包簇资源调度算法(IKPC)步骤：

步骤1 将所有待分配的聚类包放入队列，计算每个聚类的包到簇的距离，并将符合条件的聚类包映射到离它最近的簇中；

步骤2 若簇资源不够分配给任何一个聚类包，则将聚类包放入队列尾部，直到聚类包找到具有符合条件的簇进行映射；

步骤3 循环步骤1和步骤2，直到所有的经过聚类以后的包映射到相应的簇中。

4 相关实验

实验1 包聚类算法验证。

为了验证包聚类算法的有效性，现定义几种包参数，具体参数如表2所示。包需求为(100,100)的含义是对CPU的需求为100核，对内存的需求为100 MB。

表2 包需求参数(2)

Tab.2 Package demand parameters(2)

包编号	包需求
1	(100,100)
2	(300,200)
3	(600,300)
4	(500,900)
5	(200,300)
6	(400,500)
7	(800,700)
8	(1 000,800)
9	(1 000,500)
10	(1 000,900)

通过Hadoop框架的MapReduce思想进行改进的*k*-means算法的验算，实验结果如图2所示。

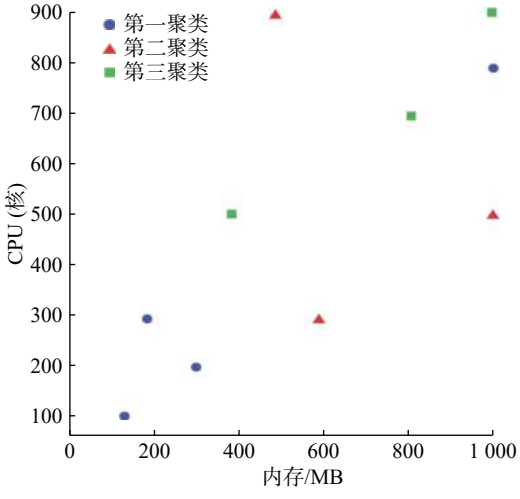

图2 实验聚类结果

Fig.2 Clustering experiment results

由图2的实验结果可得，10个包样本，经过聚类最终形成3个聚类包。第一类包编号为1，2，5，8的聚类包，编号2，8为对CPU需求较高的包，编号5为对内存需求较高的包；第二类编号为3，4，9的聚类包，编号3，9为对CPU需求较高的包，编号4为对内存需求较高的包；第三类编号为6，7，10的包，编号7，10为对CPU需求较高的包，编号6为对内存需求较高的包。因此，本实验在将包进行聚类时，具有一定的有效性和灵活性。

实验 2　资源调度算法验证。

采用云仿真软件 Cloudsim3.0[14]进行实验，实验环境中计算机参数为 Intel Core i5-8250U CPU @1.60 GHz，内存为 8.00 GB，64 位操作系统。

为了说明在资源调度算法之前首先将具有需求差异的包进行聚类的重要性，将本文的 IKPC 算法与未经聚类直接采用资源调度算法（PC，package-cluster）进行对比，此外，进行对比的算法还有传统启发式算法、降序首次适应算法（FFD）[15]，FFD 算法是通常用来解决资源调度问题的方法。为了验证在包的需求更多以及簇资源规模更大时算法的有效性，现采用 2 000 个簇和 1 600 个包进行实验。

如图 3 所示，本文中经过包聚类以后的 IKPC 算法的簇使用个数最少，相比未经过聚类，直接调用 PC 算法的簇的个数少 100 左右，相比于簇使用个数最多的 FFD 算法，IKPC 的簇使用个数少 500 左右。因此，本文中的 IKPC 能在最少的簇中尽可能多地分配资源，具有一定的可用性。

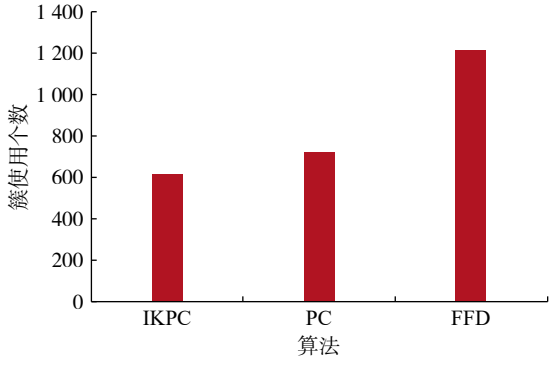

图 3　簇使用个数比较

Fig.3　Cluster number comparison

如图 4 所示，本文的资源调度算法 IKPC 的 CPU 利用率最高，FFD 算法与其他两种算法相比，其 CPU 利用率最低。IKPC 算法比 FFD 算法的 CPU 利用率总体高 10% 左右，比未聚类的 PC 高 6% 左右。显而易见，本文的资源调度算法在 CPU 利用率方面更有优势。

如图 5 所示，IKPC 的内存利用率最高，相比于内存利用率最低的 FFD 算法，IKPC 的内存利用率高了将近 15%，比未聚类的 PC 高 7% 左右。因此，本文的资源调度算法 IKPC 在提高内存利用率方面具有有效性。

图 6 为 3 种算法的能耗比较结果。由于本文的 IKPC 算法在资源分配过程中使得包能够集中映射到簇中，从而降低了资源分配过程中产生的能耗，而 PC 和 FFD 算法在资源分配过程中侧重于寻找合理的资源分配解，忽略了能耗、资源利用率的优化，因此，本文的 IKPC 算法在降低数据中心的能耗方面具有一定的可用性。

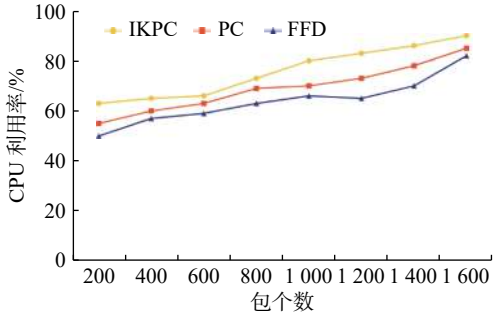

图 4　CPU 利用率比较

Fig. 4　CPU utilization comparison

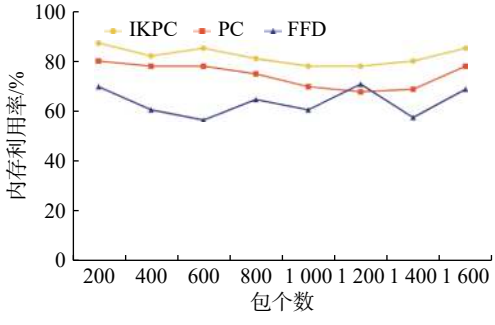

图 5　内存利用率比较

Fig. 5　Memory utilization comparison

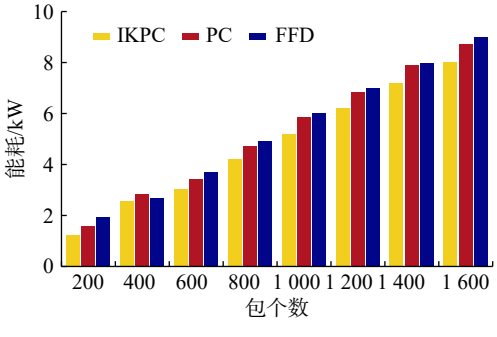

图 6　能耗比较

Fig. 6　Energy consumption comparison

5　结束语

针对数据中心包簇框架下的资源分配问题，本文在进行包簇资源映射工作之前，首先将具有资源需求差异的包采用改进的 *k*-means 进行聚类，然后，通过皮尔逊相关系数将聚类以后的包的需

求和簇可提供资源之间的相关性作为距离度量因子，实现了包簇资源的映射。实验研究表明，改进后 *k*-means 算法能够有效地对包聚类，资源调度算法能够有效地减少簇的使用个数，不仅能够提高各类资源利用率，而且可以降低数据中心由于资源分配而产生的能耗。

参考文献：

[1] HE J, ZHANG Y X, JU L, et al. Block-stream as a service: a more secure, nimble, and dynamically balanced cloud service model for ambient computing[J]. IEEE Network, 2018, 32(1): 126–132.

[2] 朱亚会, 陈丹, 庄毅. 云数据中心资源利用率均衡的虚拟机调度算法[J]. 小型微型计算机系统, 2017, 38(2): 232–237.

[3] ZHOU B Y, WU L, WANG L, et al. Joint optimization of server and network resource utilization in cloud data centers[C]//Proceedings of GLOBECOM 2017 IEEE Global Communications Conference. Singapore: IEEE, 2017: 1-6.

[4] SELIM G E I, EL-RASHIDY M A, EL-FISHAWY N A. An efficient resource utilization technique for consolidation of virtual machines in cloud computing environments[C]// Proceedings of 2016 33rd National Radio Science Conference. Aswan: IEEE, 2016: 316-324.

[5] COMBARRO M, TCHERNYKH A, KLIAZOVICH D, et al. Energy-aware scheduling with computing and data consolidation balance in 3-tier data center[C]//Proceedings of 2016 International Conference on Engineering and Telecommunication. Moscow: IEEE, 2016: 29-33.

[6] 房丙午, 黄志球. 云计算中能耗和性能感知的虚拟机优化部署算法[J]. 计算机工程与科学, 2016, 38(12): 2419–2424.

[7] 卢浩洋, 陈世平. 基于包簇映射的云计算资源分配框架[J]. 计算机应用, 2016, 36(10): 2704–2709.

[8] 王法玉, 刘志强. Spark 框架下分布式 *K*-means 算法优化方法[J]. 计算机工程与设计, 2019, 40(6): 1595–1600.

[9] TONG J F. User clustering based on Canopy+*K*-means algorithm in cloud computing[J]. Journal of Interdisciplinary Mathematics, 2017, 20(6/7): 1489–1492.

[10] 徐雨婷, 陈世平. 基于能耗感知的包簇资源分配算法研究[J]. 软件, 2019, 40(2): 141–146.

[11] 袁玉美. 一种均衡物理资源和网络带宽的虚拟机部署方法研究[J]. 无线互联科技, 2017(7): 40–41.

[12] WANG Y, XIA Y. Energy optimal VM placement in the cloud[C]//Proceedings of 2016 IEEE 9th International Conference on Cloud Computing. San Francisco: IEEE, 2016: 84-91.

[13] MENG X Q, SCI C, KEPHART J, et al. Efficient resource provisioning in compute clouds via VM multiplexing[C]// Proceedings of the 7th International Conference on Autonomic Computing. Washington, DC, USA: ACM, 2010: 11-20.

[14] 王霞俊. CloudSim 云计算仿真工具研究及应用[J]. 微型电脑应用, 2013, 29(8): 59–61.

[15] 林伟伟, 刘波, 朱良昌, 等. 基于 CSP 的能耗高效云计算资源调度模型与算法[J]. 通信学报, 2013, 34(12): 33–41.

（编辑：石　瑛）