文章编号:1671-4598(2021)12-0177-07 DOI:10.16526/j.cnki.11—4762/tp.2021.12.033 中图分类号:TP3 文献标识码:A

基于模拟退火算法的粒子群优化算法在容器调度中的应用

刘哲源[1]，吕晓丹[1]，蒋朝惠[1,2]
(1.贵州大学 计算机科学与技术学院，贵阳 550025；
2.贵州省公共大数据重点实验室，贵阳 550025)

摘要：随着互联网产业的发展，虚拟机创建速度慢、不易扩展、灵活性不足等缺点越来越凸显，容器技术的出现为这些问题提出了一种新的解决思路；而现有的调度算法仅考虑容器云集群中工作节点的内存、CPU等物理资源，没有考虑对容器云调度后的镜像分发过程有明显影响的网络负载率，导致容器调度任务等待时间过长，造成数据中心的资源浪费；鉴于粒子群优化算法在局部开采能力和全局探测方面有较强的优势，提出了一种基于模拟退火算法的粒子群优化算法（SA—PSO，simulated annealing particle swarm optimization algorithm）的容器调度算法，通过使用模拟退火优化粒子群算法使其在算法初期跳出局部最优情况，提升算法性能；在Kubernetes平台实验过程中，SA—PSO调度算法相比Kubernetes的BalancedQosPriority算法，提升了整体节点资源利用率，显著减少任务最少等待时间；同时与标准PSO算法以及动态惯性权重PSO算法进行对比，不仅收敛能力有显著提升，并且相较标准PSO算法全局最优节点命中率提升近60%。

关键词：粒子群优化算法；模拟退火；Kubernetes；Docker；调度算法

Application of Particle Swarm Optimization Algorithm Based on Simulated Annealing Algorithm in Container Scheduling

LIU Zheyuan[1], LÜ Xiaodan[1], JIANG Chaohui[1,2]
(1. College of Computer Science and Technology, Guizhou University, Guiyang 550025, China;
2. Guizhou Provincial Key Laboratory of Public Big Data, Guiyang 550025, China)

Abstract: With the development of the Internet industry, shortcomings such as slow creation of virtual machines, difficult expansion, and insufficient flexibility have become more and more prominent. The emergence of container technology has proposed a new solution to these problems. The existing scheduling algorithm only considers the memory, CPU and other physical resources of the working nodes in the container cloud cluster, and does not consider the network load rate that has a significant impact on the image distribution process after the container cloud scheduling, resulting in too long waiting time for container scheduling tasks. Cause a waste of resources in the data center. Considering that the particle swarm optimization algorithm has strong advantages in local mining capabilities and global detection, a simulated annealing algorithm—based particle swarm optimization algorithm (SA—PSO) container scheduling algorithm is proposed. Using simulated annealing to optimize the particle swarm algorithm makes it jump out of the local optimal situation in the early stage of the algorithm, and improves the performance of the algorithm. In the process of the Kubernetes platform experiment, the SA—PSO scheduling algorithm compared to Kubernetes' Balanced QosPriority algorithm improves the overall node resource utilization and significantly reduces the minimum waiting time of tasks; at the same time, it is compared with the standard PSO algorithm and the dynamic inertia weight PSO algorithm, not only the convergency ability has been significantly improved, and compared with the standard PSO algorithm, the global optimal node hit rate has increased by nearly 60%.

Keywords: particle swarm optimization algorithm; simulated annealing; Kubernetes; Docker; scheduling algorithm

0 引言

随着云计算产业的高速发展，面对高并发量的业务压力，虚拟机灵活性不足、创建速度慢、不易扩展等缺点越来越凸显。容器技术易扩展、秒级启动、灵活的弹性伸缩等特点，为这些问题提供了一种全新的解决方案。软件容器化已经成为软件部署和当今互联网的新趋势。Docker[1]则

收稿日期:2021-04-29； **修回日期**:2021-05-20。
基金项目:贵州省科技计划资助项目(黔科合基础[2017]1051)。
作者简介:刘哲源(1992-)，男，江苏徐州人，硕士研究生在读，主要从事云计算和容器技术方向的研究。
吕晓丹(1970-)，男，上海人，副教授，硕士生导师，主要从事云计算和数据分析方向的研究。
蒋朝惠(1965-)，男，四川广安人，教授，硕士生导师，主要从事网络安全和系统安全方向的研究。
引用格式:刘哲源，吕晓丹，蒋朝惠.基于模拟退火算法的粒子群优化算法在容器调度中的应用[J].计算机测量与控制，2021，29(12)：177-183.

是其中一个开源的应用容器引擎，不仅是一个新型虚拟化技术，同时也是应用程序交付的一种方式。传统虚拟机一般由虚拟硬件和操作系统两部分组成，而 Docker 容器不需要安装特定的操作系统和 VMM（虚拟机监控），Docker 容器通过共享主机操作系统内核，用主机的硬件和操作系统来完成程序的操作，一定程度上减少了资源的浪费问题，两者架构对比如图 1 所示。

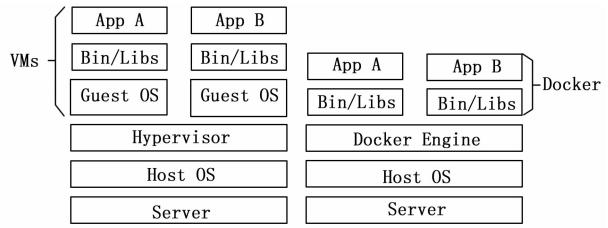

图 1 传统 VM 和 Docker 的架构对比

Kubernetes[2] 是目前主流的 Docker 容器编排平台之一，负责对容器的整个生命周期的管理。Pod 是 Kubernetes 的基本运行单元，一个 Pod 内可以有多个容器，且可以在创建 Pod 的 yaml 配置文件中声名容器所需要的资源需求。Kubernetes 在对 Pod 任务进行调度时，首先通过预选算法筛选出所有符合运行条件的节点，接着通过优选算法为这些节点进行打分，最终选择分数最高的节点用来绑定 Pod 任务，接着调用 Docker－daemon 通过 Http 方式向镜像仓库下载相应的容器镜像，具体步骤如图 2 所示。

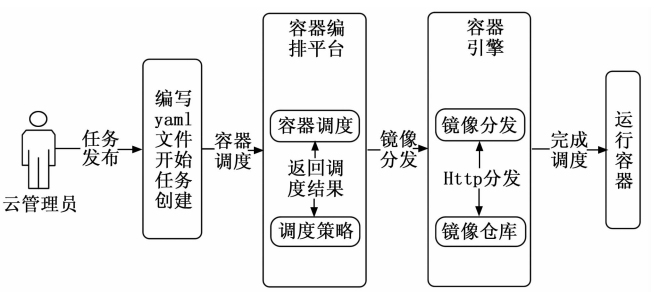

图 2 Kubernetes 调度过程

虽然 Kubernetes 的 Balanced Resource Allocation 调度算法虽然能一定程度上解决资源均衡问题，但是仍然达不到数据中心的预期水平。Kubernetes 的调度算法仅仅考虑了工作节点的 CPU 和内存利用率，而未考虑工作节点的网络负载能力，从而影响 Pod 的任务最少等待时间。

目前，国内外对容器调度后的资源利用率不均衡问题有较多的研究。文献［3］设计了 EVCD（基于虚拟机弹性供给的容器部署），利用贪婪 first－fit 和 round－robin，并 EVCD 可以在运行时重新分配容器实现 QoS 的改进，提升容器调度的效果。文献［4－5］通过对容器需求的 CPU 和内存资源进行分类，针对不同类型的容器使用不同的调度算法。文献［6］提出了基于应用历史记录的调度算法，减少容器调度后的镜像分发过程中带来的网络资源的损耗。文献[7－10]使用群优化算法对容器的多指标调度设计，实现了较好的资源使用率。文献［11］通过对数据中心资源进行更细粒度的划分，在保证了服务器性能的同时，集群资源利用率也得到了提高。文献［12－13］针对容器动态调度的问题，设计寻找最优解的动态调度算法，解决了动态调度中的资源浪费问题。文献［14］设计一种多目标容器调度算法 Multiopt，通过对工作节点当前状态进行计算评估出合适的调度节点。文献［15－16］对集群负载均衡性进行考虑，采用智能算法对集群中所有节点进行评定，最终确定调度节点。文献［17］通过预先训练随机森林回归模型，预测集群压力进行容器调度。文献［18］提出了一种 VM－Container 混合分层资源调度机制，通过调度状态综合性将任务划分为不同的级别，针对不同的级别制定不同的 PSO 调度算法，但没有考虑到特殊情景下的容器调度。文献［19－20］关注容器资源要求环境和任务规模变化，提出采用粒子群优化算法来进行容器部署，解决了容器部署过程中的网络消耗问题。

尽管国内外很多学者已经为此做了很多有价值的工作，但是仍然存在不足。因此，研究的重点在于改进容器调度算法以提高集群节点的资源利用率从而达到数据中心的预期水平，减少容器任务最少等待时间，提升容器云的服务质量。

1 数学模型的设计与构建

在 Kubernetes 平台中，创建容器首先采用声明式方法在 Pod 的 yaml 文件中声明容器所需要的资源信息，接着通过 kubectl 命令来创建 Pod。因此，在 Pod 开始调度之前，系统已经获取了 Pod 的资源需求情况。同时，Kubernetes 中的 api－server 组件收集集群中节点的指标资源使用情况。构建数学模型如下。

容器云集群：

$$Cluster = \{Node1, Node2, \ldots, Noden\} \tag{1}$$

式中，$Cluster$ 为 n 个 $Node$ 节点组成的集群，每个 $Nodei$ 节点都具有 4 维属性表示为：

$$Nodei = \{U_{CPU}, U_{Mem}, U_{Net}, status\_now\} \tag{2}$$

式中，$Nodei^d$ 表示为 $Node$ 节点 i 中的可用第 d 维资源，$Nodei^4$ 表示节点 $Nodei$ 得可用第 4 维资源 $status\_now$，即为当前主机状态，其具体取值如下：

$$status\_now = \begin{cases} 1, \text{if} & Node_i \quad \text{on} \\ 0, \text{if} & Node_i \quad \text{down} \end{cases} \tag{3}$$

式中，当节点 $Nodei$ 当前处于宕机状态时，$status\_now$ 为值 0，反之，$status\_now$ 为值 1。

对于每次创建任务 Pod_i，都具备 3 维属性：

$$Pod_i = \{R_{\text{CPU}}, R_{\text{Mem}}, R_{\text{image}}\} \tag{4}$$

式中，R_{mem}，R_{image} 分别为任务 Pod_i 所需要的 CPU、内存、容器镜像等资源需求。当开始调度时，通过 Kubernetes 编排平台中的预选决策后，集群已经通过预选决策筛选出了一部分可以满足调度任务可正常工作节点：

$$\begin{cases} Node = \{Nodei, \ldots, Nodek\}, i, k \in [1, n], \\ Node_z = \{U_{\text{CPU}}, U_{\text{mem}}, \ldots\}, Node_z \in Node \end{cases} \tag{5}$$

式中，*Node* 为筛选出来符合调度条件的工作节点的集合，且被筛选出来的 *Node* 节点均包含有当前节点目前的资源使用情况，U_{CPU}为节点当前的内存使用量，U_{mem}为当前节点的内存使用量。尽管筛选出了一部分可以使用的工作节点，我们仍需要通过优选测了为 Pod_i 找到一个最优节点进行调度，以提升集群的整体资源利用率。

对于每个数据中心而言，CPU、内存等资源的长期闲置也是一种开销损耗，而资源利用率超过一定负荷阈值则证明其在负荷运行，同样也是对服务器的生命周期的一种损耗，因此保证资源利用率不低于资源阈值，且不高于负荷阈值，则是提升资源利用率的一种有效方式。对于资源利用率我们希望其：

$$\begin{cases} Z_{\min} \leqslant Z_{Nodei-use} \leqslant Z_{\max} \\ Z_{Nodei-use} = \dfrac{Z_{Nodei-U} + Z_{Podi-R}}{Z_{Nodei-t}} \\ Z = \{CPU, Mem, \ldots\} \end{cases} \tag{6}$$

式中，Z 表示为资源类型，包括 CPU、内存、网络等资源类型，$Z_{\min}$ 为云服务厂商所希望资源不低于的最低资源阈值，$Z_{\max}$ 为云服务厂商所希望的不高于的负荷阈值。$Z_{Nodei-use}$ 为若 Pod_i 绑定到 *Nodei* 节点后 Z 资源使用率，$Z_{Nodei-t}$ 为 *Nodei* 节点的总资源，$Z_{Nodei-U}$ 为当前节点此时刻的 Z 资源类型的资源使用量，Z_{Podi-R} 为任务 Pod_i 所需要的资源量。

容器云调度过程应尽可能多地为上层应用分配计算机资源，给容器云经营者带来更多的收益，因此，以所有执行任务所需要的资源量的价值来表示容器云经营者满意度函数如式（7）所示：

$$f(Z, Nodei) = (Z_{\min} + Z_{\max}) * Z_{Nodei-t} - 2U_Z \tag{7}$$

式（7）意为：当适应性函数在节点 i 的函数值越高，则证明资源 Z 在 *Nodei* 上的资源使用率低，越不符合数据中心的资源使用率预期，因此是 *Pod* 任务绑定的较优节点。在 *Pod* 任务绑定到 *Nodei* 节点后，Z 资源的使用率高于最低资源阈值，且不高于最高负载阈值，以达到容器云运营者的满意程度。

因此，我们主要对数据中心资源使用不均情况较为常见的 CPU、内存指标和工作节点当前网络负载情况进行数学建模：

1）CPU 模型构建。根据 *Pod* 所需要的 CPU 需求量 R_{CPU}，与筛选出的节点 *Nodei* 中目前所使用的 CPU 使用量 U_{CPU} 进行量化建模，具体公式如式（8）所示：

$$\begin{cases} E_1 = E_{CPU}(Pod, Nodei) \\ E_{CPU} = R_{CPU} * f(CPU, Nodei) \end{cases} \tag{8}$$

为了满足任务 Pod 所需 CPU 资源，E_1 应受限于：

$$0 < Cpu_{\min} < Cpu_{\max} < 1$$

2）内存模型构建。根据 *Pod* 所需要的内存需求量 R_{Mem}，与筛选出的节点 *Nodei* 中目前所使用的内存使用量 U_{Mem} 进行量化建模，具体公式如式（9）所示：

$$\begin{cases} E_2 = E_{Mem}(Pod, Nodei) \\ E_{Mem} = R_{Mem} * f(Mem, Nodei) \end{cases} \tag{9}$$

为了满足任务 *Pod* 所需内存资源，E_2 应受限于：

$$0 < Mem_{\min} < Mem_{\max} < 1$$

3）网络负载建模。根据 *Pod* 所需要的镜像需求 R_{image}，与筛选出的节点 *Pod* 中目前所使用的网络负载量 U_{net} 进行量化建模，具体公式如式（10）所示：

$$\begin{cases} E_3 = E_{net}(Pod, Node_i) \\ E_{net} = R_{image} * (1 - U_{net}) \end{cases} \tag{10}$$

4）适应性函数构建。针对以上问题，对适应性函数公式可以建立为：

$$function(Pod, Nodei) = \sum_{i=1}^{n} E_i * Nodei^4 \tag{11}$$

式（11）中，$Nodei^4$ 是 *Nodei* 得第 4 维属性 *status _ now*，当节点宕机时，该宕机节点得适应性函数 *function* 值为 0，当适应性函数 *function* 值越高，则代表当前 *Nodei* 资源利用率越满足任务 *Pod* 运行条件，且绑定 *Pod* 任务后集群资源利用率趋于数据中心满意度。

2　基于模拟退火的粒子群优化算法

2.1　标准粒子群优化算法

粒子群优化算法（PSO）[21]是一种模拟生物行为活动的群智能化算法，每个粒子都存在一个存在可行解的搜索空间，且每个粒子都具备一定的空间搜索能力，通过粒子速度 V，粒子位置 X 来表示粒子空间位置特征，并由目的函数确定适应度值，其值的好坏表示粒子的优劣程度。

粒子在独自的搜索空间内，每个粒子会记录自身最优位置与全局最优位置，并通过判断当前自身位置和个体最优位置以及群体最优位置的距离，若自身位置优于全局最优位置，则更新全局最优位置，并不断更新自身的位置和速度，不断迭代，直到找到种群最优位置。更新公式如式（12）所示：

$$\begin{cases} V_{id}^{k+1} = \omega V_{id}^{k} + c_1 r_1 (P_{\mathrm{best}_{id}} - X_{id}^{k}) + c_2 r_2 (G_{\mathrm{best}_d} - X_{id}^{k}), \\ X_{id}^{k+1} = X_{id}^{k} + V_{id}^{k+1}, \\ i \in pN \end{cases} \tag{12}$$

式（12）中，pN 代表粒子群大小，X_{id}^{k} 代表粒子 i 在 d 维空间内迭代至 k 代后的当前位置，V_{id}^{k} 代表粒子 i 在 d 维空间内迭代至 k 代后的当前速度，$P_{\mathrm{best}_{id}}^{k}$ 代表粒子 i 在 d 维空间内的个体最优位置，G_{best_d} 代表粒子群在 d 维空间内的群体最优位置。ω 为惯性权重，用于调节对解空间的搜索范围。在式（12）中的粒子速度的更新公式中，主要包含 3 个部分：

1）粒子 i 先前的速度 V_{id}^{k}；

2）个体认知部分，表示为粒子本身的思考，可理解为粒子 i 当前的位置 X_{id}^{k} 与自己最好位置 $P_{\mathrm{best}_{id}^{k}}$ 之间的距离；

3）社会认知部分，表示粒子间的信息共享与合作，可以理解为粒子 i 当前位置 X_{id}^{k} 与群体最好位置 G_{best_d} 之间的距离。

标准 PSO 算法的伪代码如下所示：

Algorithm 1：Find the maximum position of the maximum func-

tion.

procedure PSO

for each particle i

Initialize velocity V_i and position X_i for particle i

Evaluate particle i and set $p_{\text{Best}_i} = X_i$

end for

$g_{\text{Best}} = \max\{ p_{\text{Best}_i} \}$

while not stop

for $i = 1$ to N

Update the velocity and position of particle i

Evaluate particle i

if function(X_i) > function(p_{Best_i})

$p_{\text{Best}_i} = X_i$;

if function(p_{Best_i}) > function(g_{Best})

$g_{\text{Best}} = p_{\text{Best}_i}$;

end for

end while

print g_{Best}

end procedure

2.2 基于模拟退火的粒子群优化算法思想

模拟退火算法[22]的是受固体退火过程的启发而发展出来的一种算法，固体加热至足够的高的温度时，再另其缓慢冷却，而固体中的粒子在升温过程中，内能增大，趋于无序；缓慢降温时又能够使得内能减小，趋于有序。因此理论上说，只要初始温度足够高，而降温过程又足够缓慢，那么算法在过程中将会收敛到全局极值。而之所以模拟退火算法可以有能力进行概率突跳，是因为采用了 Metropolis 准则，其表达式为如式（13）所示：

$$P_{ij}^{T} = \begin{cases} 1, & E(i) \geqslant E(j) \\ e^{-E(j)-E(i)/KT}, & E(i) < E(j) \end{cases} \tag{13}$$

式中，$E(i)$ 和 $E(j)$ 分别为固体在状态 i 和 j 下的内能，K 为玻尔兹曼常数，T 为当前温度，P 为 T 温度下，系统接受状态到 i 状态 j 的新状态的概率。因此由式可知，若 $E(i) < E(j)$，则接受新的状态 j，反之，则以概率 $P_{ij}^{T} = e^{-E(j)-E(i)/KT}$ 接受新状态 j。

如图 3 所示，在主节点 api－server 组件接收到 Pod 任务创建任务后，kube－scheduler 组件调用基于模拟退火的粒子群优化算法，并向数据库请求实时采集容器云中工作节点的当前资源信息，得到返回数据后开始计算，计算完成后返回最优节点，工作节点 kubelet 组件通过 watch 监听主节点 api－server 得到调度任务后，调用本地 Docker－daemon 组件进行容器镜像的检索和下载工作，镜像存在本机后，启动任务，至此一次任务创建完成。

粒子群优化算法的优化过程主要是通过粒子群中的粒子通过粒子间的信息（即个体最优解和全局最优解）不断更新自身的位置和速度，使其能不断地向全局最优解靠拢，最终得到全局最优解。但是由于基本粒子群算法处于运算后期时，因粒子多样性减少，容易陷入局部最优，且后期

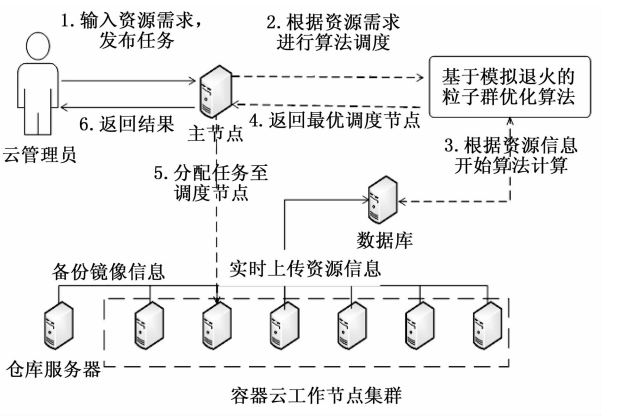

图 3 基于模拟退火的粒子群优化算法框架图

收敛速度变慢。在整个计算流程中，惯性权重对粒子速度和位置的影响不同：运算前期，较大的惯性权重有利于全局搜索，粒子速度快，但寻优值不够精确；相反，运算前期，较小的惯性权重有利于局部搜索，全局搜索能力较弱，但容易陷入局部最优。因此，将模拟退火算法嵌入粒子群优化算法中，就是在算法初期温度较高的时候，让种群例子有较大概率去接受非最优解，从而跳出局部最优解，而后期温度降低时，又能够收敛到全局最优，准确定位到全局最优解。由于模拟退火有能够突跳的概率，因此可以有效避免算法陷入局部最优，有效避免了“早熟”现象。基于模拟退火的粒子群优化算法流程如图 4 所示。

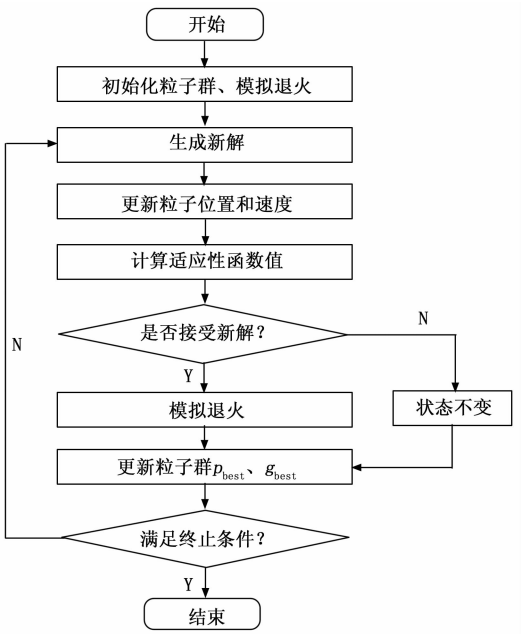

图 4 模拟退火的粒子群优化算法流程图

3 实验结果与分析

3.1 实验环境

实验环境采用翔明云提供的虚拟机作为操作主机，创建集群大小为 17 的高可用 Kubernetes 集群，集群中 3 台节

点搭建 Master 节点高可用，其中 2 台既是 Master 节点也是 Node 节点，共计 16 个 Node 节点。实验环境如表 2 所示。

表 2　实验环境表

名称	版本	名称	版本
操作系统	CentOS 7.5	Go	V1.15
Kubernetes	V1.18.10	Docker	V1.19.03
CPU	4	内存	4 GB
磁盘	500G	带宽	100 Mbps

对 SA－PSO 算法（后称改进 PSO 算法）与标准 PSO 算法、文献［23］动态权重调整的 PSO 算法（后称权重 PSO 算法）进行性能及精度进行对比。同时对改进 PSO 算法与 Kubernetes 默认调度算法进行对比。服务器资源利用率的期望阈值根据贵州翔明云日常运维需求来进行设定，当 CPU 利用率在 20%～80%，内存利用率在 30%～80%之间，服务器性能较好，且资源利用率最合理，文献［24］实验得出当 $c_1 = c_2 = 2.5$ 时标准 PSO 算法的收敛效果最好，文献［25］指出模拟退火中 $K = 0.7$ 和 $T_0 = 10\,000$ 时，基于模拟退火的粒子群优化算法效果较好。实验中具体参数设置如表 3 所示。

表 3　实验参数表

参数名称	参数值	参数描述
pN	3	粒子群中粒子个数。
max_*iter*	50	粒子群最大迭代次数。
$\omega_{\min}$	0.4	粒子群最小惯性权重。
$\omega_{\max}$	0.9	粒子群最大惯性权重。
c_1	2.5	粒子群中个体学习因子。
c_2	2.5	粒子群全局学习因子。
$cpu_{\min}$	20%	期望工作节点 cpu 利用率的最小阈值。
$cpu_{\max}$	80%	期望工作节点 cpu 利用率的最大阈值。
$mem_{\min}$	30%	期望工作节点内存利用率的最小阈值。
mem_{max}	80%	期望工作节点内存利用率的最大阈值。
K	0.7	模拟退火中玻尔兹曼常数。
T_0	10 000	模拟退火算法中的初始温度。

3.2　实验与分析

3.2.1　性能测试和精度测试

对改进 PSO 算法的性能进行测试，通过对权重 PSO 算法，标准 PSO 算法的寻优过程中的迭代次数以及收敛性进行对比，测试改进算法的算法效率。实验过程通过发布一个 CPU 需求量为 1 000 m、内存需求量为 1 G 的 Nginx 容器 Pod 任务，使用目标函数对不同 Node 节点的资源利用情况进行计算得到该节点的适应性值，适应值最高的节点即为本次任务的全局最优目标节点，不同的粒子在不同节点间自主运动，并不断与全局最优解进行比较，调整自身的运动方向以及运动速度，直到找到最高峰值，峰值所在节点既是全局最优节点，具体如图 5 所示。

当迭代次数为 50，粒子群大小为 3 时，对 3 种算法的迭代寻优变化曲线进行对比，具体实验结果如图 6 所示。

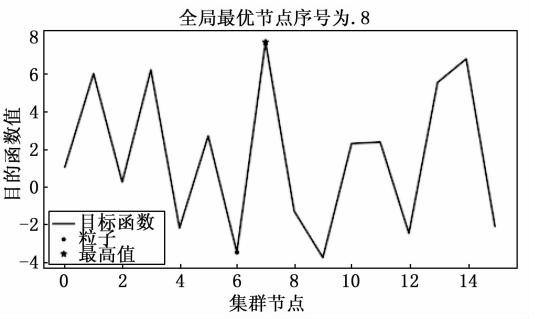

图 5　集群节点适应值曲线

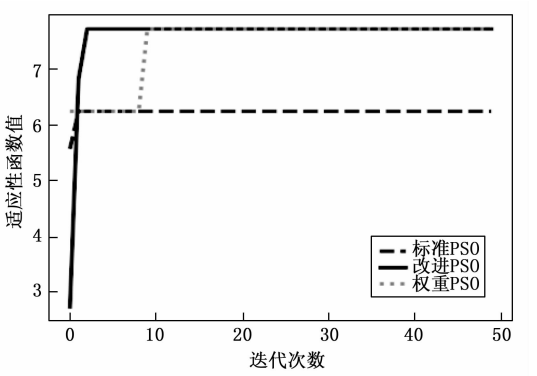

图 6　算法迭代寻优对比图

依此反复测试 10 次，目的函数达到最大值记为一次成功命中，意味着没有进入局部最优，达到了全局最优解，记为全局收敛成功。各个算法寻找到各自群体最优解看作完成迭代，记录为迭代次数，同时该记录也包含陷入局部最优解的情况。通过计算得出平均迭代次数、收敛成功率情况如表 4 所示。

表 4　寻优情况表

算法名称	命中次数	收敛成功率/%	平均迭代次数
标准 PSO	3	30	39
权重 PSO	7	70	22
改进 PSO	9	90	18

实验表明，当粒子群大小为 3，最大迭代次数为 50 次时，标准 PSO 算法容易陷入局部最优情况而无法精确地找到目标节点，而改进 PSO 算法和权重 PSO 算法相较标准 PSO 算法的收敛成功率以及命中率均有较大提升，且平均迭代次数要少于标准 PSO 算法。改进 PSO 算法由于对学习因子进行动态调整，收敛性能略优于权重 PSO 算法。上述实验可知，在本次实验环境下，改进 PSO 算法的成功命中率相较标准 PSO 算法提高了近 60%。

3.2.2　负载均衡测试

对集群负载均衡度进行测试。使用 Kubernetes 的 Scheduler－framework 用来实现改进 PSO 算法的调度算法。通过选择先后选择 BLA 调度算法和改进 PSO 调度算法分别部署 4 个高资源需要求的 Nginx 服务器作为调度任务，在

yaml 文件中限制 CPU 大小为 1 000 m，内存大小为 1 G，通过 Kubernetes 的 BalanceResourceAllocation（后称 BLA）算法，将 4 个调度任务分别绑定到 4 个节点，先后次序为 node15，node02，node05，node12，完成后集群资源利用使用图如图 7 所示。

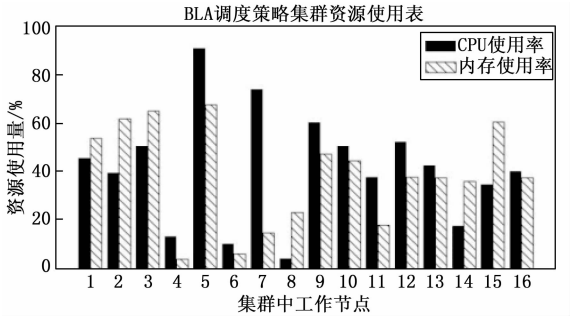

图 7 BLA 调度算法集群资源使用情况图

实验结果表明，虽然 BLA 调度算法一定程度上提高了资源利用率，但是仍有部分节点资源使用情况不理想。在 BLA 调度过程中，主要会考虑任务绑定到某个节点时节点资源使用率是否均衡而不会考虑资源利用率情况，因此容易造成某些节点资源利用率不高的情况。通过改进 PSO 算法先后部署同样的 4 个调度任务，任务绑定节点先后次序为 node08，node06，node04，node15，改进 PSO 算法能较好的提高整体资源利用情况，达到云提供商的资源使用预期，完成调度后的资源使用情况如图 8 所示。

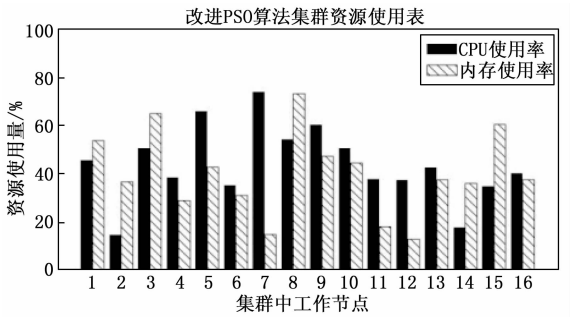

图 8 改进 PSO 调度算法集群资源使用情况图

通过对比两个算法的任务调度可以看出，改进 PSO 算法 SA－PSO 相对 Kubernetes 的 Balance Resource Allocation 算法实现了更好的集群资源利用率，同时部署节点的网络负载率也处于较低水平，为下一步容器镜像分发步骤节省了任务等待时间。在本实验环境中，改进 PSO 算法（SA－PSO）可以准确地找到任务绑定的最优节点，从而提高集群的整体资源利用率。

3.2.3 任务最少等待时间测试

使用 Kubernetes 默认调度算法和改进 PSO 调度算法进行分别部署 10 次镜像大小为 133 MB 的 Nginx 服务 Pod 任务，其资源需求配置如测试（2）相同，记录其任务最短等待时间，实验对比结果如图 9 所示。

图 9 中，网络负载率 1 为默认算法调度工作节点的网络负载率，网络负载率 2 为改进 PSO 算法调度工作节点的网络负载率。由实验结果可知，改进 PSO 算法较于默认调度算法由于在调度过程中将工作节点的网络负载率带入计算，优先选择网络负载率低的节点进行调度，因此可以提升镜像分发的效率，从而减少任务最少等待时间。

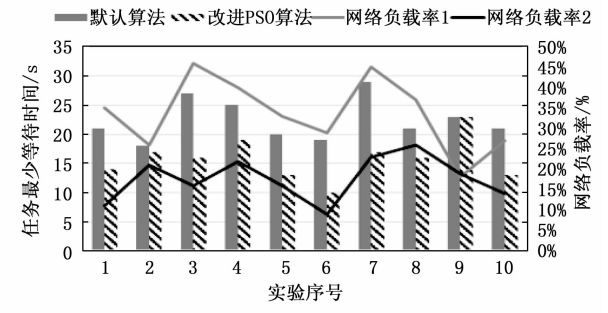

图 9 任务最少等待时间与网络负载率对比图

4 结束语

随着云计算产业的快速发展，Docker 的出现给了互联网企业一种新的解决问题的思路。面对大规模的软件部署，若不能合理利用服务器资源，无论对云提供商还是软件方都是一项额外的开销。而主流的容器编排平台 Kubernetes 在调度容器的过程中存在着资源利用不足的问题，同时目前主流研究没有将容器调度过程与镜像分发过程相结合。因此，在此提出了一种基于 SA－PSO 调度算法，通过模拟退火算法解决粒子群算法易于陷入局部最优解的问题。经过实验表明，SA－PSO 算法相较标准 PSO 算法以及权重 PSO 算法有较强的收敛能力以及寻优精准度，在本实验环境中，寻优精准度相较标准 PSO 算法提升约 60%。同时，在 Kubernetes 平台实验测试中，SA－PSO 调度算法不仅对数据中心期望的资源阈值进行考虑，而且在集群资源利用率方面优于 BalanceResourceAllocation 调度算法。对工作节点的网络负载率进行考虑，显著减少了容器任务的任务等待时间。

参考文献：

[1] ANDERSON，CHARLES. Docker［Software engineering］［J］. IEEE Software，2015，32（3）：102－103.

[2] WILKES，JOHN，BREWER，et al. Borg，Omega，and Kubernetes［J］. Communications of the ACM，2016，59：50－57.

[3] NARDELLI M，HOCHREINER C，SCHULTERS. Elastic provisioning of virtual machines for container deployment［C］// Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. ACM，2017：5－10.

[4] ROSA R L，PIEGAS K G，Marcelo P，et al. Time－constrained and network－aware containers scheduling in GPU era［J］. Future Generation Computer Systems，2021，117：72－86.

[5] CANOSA R，TCHERNYKH A，CORTES－MENDOZA J M，et，al. Energy consumption and quality of service optimization

in containerized cloud computing [C] //2018 IVANNIKOV Ispras Open Conference (ISPRAS), IEEE, 2018: 47 - 55.

[6] 何　龙，刘晓洁. 一种基于应用历史记录的 Kubernetes 调度算法 [J]. 数据通信，2019，3：33 - 36.

[7] LI J, LIU B, LIN W, et, al. An improved container scheduling algorithm based on PSO for big data applications [C] //International Symposium on Cyberspace Safety and Security. Springer, Cham, 2019: 516 - 530.

[8] LIN M, XI J, BAI W, et, al. Ant colony algorithm for multi - objective optimization of container－based microservice scheduling in cloud [J]. IEEE Access, 2019 (99): 1 - 1.

[9] UNGUREANU O M, VLADEANU C, KOOIJ R. Kubernetes cluster optimization using hybrid shared － state scheduling framework [C] //Proceedings of the 3rd International Conference on Future Networks and Distributed Systems, 2019: 1 - 12.

[10] KAEWKASI C, CHUENMUNEEWONG K. Improvement of container scheduling for Docker using ant colony optimization [C] // International Conference on Knowledge & Smart Technology, IEEE, 2017: 254 - 259.

[11] 张可颖，彭丽苹，吕晓丹，等. 开源云上的 Kubernetes 弹性调度 [J]. 计算机技术与发展，2019，29 (2)：109 - 114.

[12] BUYYA R, RODRIGUEZ M A. TOOSI A N, et al. Cost－efficient orchestration of containers in clouds: a vision, architectural elements, and future directions [J]. Journal of Physics: Conference Series, 2018, 1108: 012001.

[13] 左　灿，刘晓洁. 一种改进的 Kubernetes 动态资源调度方法 [J]. 数据通信，2019，2：50 - 54.

[14] LIU B, LI P F, LIN W W, et al. A new container scheduling algorithm based on multi－objective optimization [J]. Soft Computing, 2018, 22: 7741 - 7752.

[15] 林伟伟，王泽涛. 基于遗传算法的 Docker 集群调度算法 [J]. 华南理工大学学报（自然科学版），2018，046 (003)：127 - 133.

[16] 李东光，刘智平，姜雨菲. 蚁群优化算法的 Docker 集群调度算法 [J]. 西安工业大学学报，2019 (3)：330 - 335.

[17] LYU J, WEI M, YU Y. A container scheduling algorithm based on machine learning in microservice architecture [C] // 2019 IEEE International Conference on Services Computing (SCC), IEEE, 2019.

[18] FAN C, WANG Y, WEN Z. Research on improved 2D－BP-SO－based VM－container hybrid hierarchical cloud resource scheduling mechanism [C] // 2016 IEEE International Conference on Computer and Information Technology (CIT), IEEE, 2016.

[19] WANG B, WANG C, SONG Y, et, al. A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds [J]. Cluster Computing, 2020 (7): 1 - 26.

[20] WU L, XIA H. Particle swarm optimization algorithm for container deployment [J]. Journal of Physics, Conference Series, 2020, 1544: 012020.

[21] KENNEDY J, EBERHART R. Particle swarm optimization [C] // ICNN95 － international Conference on Neural Networks, IEEE, 2002.

[22] HWANG C R. Simulated annealing: Theory and applications [J]. ACTA Applicandae Mathematica, 1988, 12 (1): 108 - 111.

[23] SHI Y. A modified particle swarm optimizer [C] // Proc. of IEEE ICEC Conference, 1998.

[24] CLERC M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization [C] // Congress on Evolutionary Computation－CEC. IEEE, 2002.

[25] 李淑香. 基于模拟退火的粒子群算法在函数优化中的应用 [J]. 沈阳工业大学学报，2019，41 (6)：664 - 668.

（上接第 171 页）

[12] LI J, PAN Q. Solving the large－scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm [J]. Information Sciences, 2015, 316: 487 - 502.

[13] RAO R V, SAVSANI V J, VAKHARIA D P. Teaching－learning－based optimization: a novel method for constrained mechanical design optimization problems [J]. Computer - Aided Design, 2011, 43 (3): 303 - 315.

[14] RAO R V, SAVSANI V J, VAKHARIA D P. Teaching－learning－based optimization: an optimization method for continuous non－linear large scale problems [J]. Information sciences, 2012, 183 (1): 1 - 15.

[15] RAO R, PATEL V. An elitist teaching－learning－based optimization algorithm for solving complex constrained optimization problems [J]. International Journal of Industrial Engineering Computations, 2012, 3 (4): 535 - 560.

[16] 赵乃刚. 求解无约束优化问题的改进教与学优化算法 [J]. 小型微型计算机系统，2017，38 (9)：2107 - 2112.

[17] 马文强，张超勇，唐秋华，等. 基于混合教与学优化算法的炼钢连铸调度 [J]. 计算机集成制造系统，2015，21 (5)：1271 - 1278.

[18] 何雨洁，钱　斌，胡　蓉. 混合离散教与学算法求解复杂并行机调度问题 [J]. 自动化学报，2020，46 (4)：805 - 819.

[19] SHI F, ZHAO S, MENG Y. Hybrid algorithm based on improved extended shifting bottleneck procedure and GA for assembly job shop scheduling problem [J]. International Journal of Production Research, 2020, 58 (9): 2604 - 2625.

[20] ROSA B F, SOUZA M J F, de SOUZA S R. Algorithms based on VNS for solving the Single Machine Scheduling Problem with Earliness and Tardiness Penalties [J]. Electronic Notes in Discrete Mathematics, 2018, 66: 47 - 54.

[21] BENJAORAN V, DAWOOD N, HOBBS B. Flowshop scheduling model for bespoke precast concrete production planning [J]. Construction Management & Economics, 2005, 23 (1): 93 - 105.