DOI:10.19479/j.2095-719x.2403213

基于 Erdős-Rényi 模型的容器组合调度策略

张秀峰,巨 昊,刘 毅,李国燕
(天津城建大学 计算机与信息工程学院,天津 300384)

摘 要:容器是一种新型的虚拟化技术,相比传统虚拟机具有成本低廉、资源利用率高、利于迁移等优点.然而,现有的容器集群在调度时,容易忽略容器之间的内在关联性,只是简单地将其视为静态的个体,这会导致集群性能的失衡以及能耗的上升.针对这一问题,提出了一种基于 Erdős-Rényi 模型的容器调度策略.首先通过引入通信指标来衡量容器间的关联性;其次基于改进的 Erdős-Rényi 模型,依据关联程度对容器进行分组;最后根据分组结果,将一个 Pod 组视为一个整体进行调度,最大程度地保留容器间的关联性,从而使集群整体动态性能更加显著.仿真实验结果表明,相较于 Spread 算法,笔者提出的调度策略在任务的完成时间上降低了约 22%,能耗降低了约 5%~30%.因此,该策略模型可以帮助企业提高生产效率以及降低成本.

关键词:容器技术; Kubernetes 调度; Erdős-Rényi 模型; Cloudsim 仿真
中图分类号:TP391 **文献标志码**:A **文章编号**:2095-719X(2024)03-0213-07

Container Composition Scheduling Strategy Based on Erdős-Rényi Model

ZHANG Xiufeng, JU Hao, LIU Yi, LI Guoyan
(School of Computer and Information Engineering, TCU, Tianjin 300384, China)

Abstract: Containers are a new type of virtualization technology. Compared with traditional virtual machines, containers have the advantages of low cost, high resource utilization, and easy migration. However, when scheduling existing container clusters, it is easy to ignore the internal correlation among containers, and simply treat them as static individuals, which will lead to an imbalance in cluster performance and an increase in energy consumption. To solve this problem, this paper proposes a container scheduling strategy based on the Erdős-Rényi model. Firstly, the correlation among containers is measured by introducing communication indicators; secondly, based on the improved Erdős-Rényi model, containers are grouped according to the degree of correlation; finally, according to the grouping results, a Pod group is considered as a whole for scheduling to maximize the retention. The correlation among containers makes the overall dynamic performance of the cluster more significant. Simulation results show that compared with the Spread algorithm, the scheduling strategy proposed in this paper reduces the task completion time by about 22%, and reduces energy consumption by about 5% to 30%. Therefore, the strategy model proposed in this paper can effectively help enterprises improve production efficiency and reduce costs.

Key words: container technology; Kubernetes scheduling; Erdős-Rényi model; Cloudsim simulation

随着计算机硬件的快速迭代升级,其性能得到了飞速的提升,但在实际生产环境中仍无法实现对其的充分利用,资源过剩成为目前面临的一个主要问题.在这种背景下,云计算技术应运而生[1],而虚拟化技术作为云计算技术的核心,近年来也成为互联网领域的一大热点.容器技术出现之前,传统的虚拟化方案指的是虚拟机技术,但这种全虚拟化的方案面临着资源利用率低下、性能开销大、运维成本高昂等一系列问题.容器技术凭借其轻量级的特点,在一定程度上很好地解决了传统虚拟机技术所面临的一系列难题[2].而在实际生产环境中,随着容器数量越来越多,如何高效管理、协调、部署、监控这些容器,也成了一个难题.在这一背景下,容器编排系统应运而生.在众多的容器编排系统中,Google 推出的 Kubernetes 凭借其优秀的架构设计和完善的生态链脱颖而出,成为目前最受欢迎的容器编排系统.Kubernetes 抽象了数据中心

收稿日期:2022-11-30;**修订日期**:2023-04-03
基金项目:天津市教委科研计划项目(2016CJ12)
作者简介:张秀峰(1979—),男,山西太原人,天津城建大学讲师,博士.

的硬件基础设施,使得对外暴露的只是一个巨大的资源池,它使得企业在部署和运行组件时,不用关注底层的服务器,并实现了容器的自动化管理[3].

Pod 是 Kubernetes 中资源调度和资源分割的最小单位，由一个或多个容器组成. 在实际生产环境中的 Kubernetes 集群中,Pod 组成的网络结构复杂,包含的信息种类繁多. 而 Kubernetes 内置的调度策略为简单的“预选+优选”策略,根据 CPU 和内存两个指标对节点进行评分,调度器会将 Pod 调度至评分最高的节点之上. 但 Kubernetes 的调度策略存在一定的局限性,其在进行资源调度时只考虑了 CPU 和内存两个维度,而网络带宽、通信效率等对于集群性能也有着很大的影响. 其次,Kubernetes 的默认调度算法为 Spread 算法[4],该算法会尽可能地实现集群的负载均衡,即该算法会将工作负载平均地分布到集群各个节点上. 这会导致集群的资源碎片化,同时还会引起能耗的提高以及集群整体性能的下降.

针对 Kubernetes 中 Spread 调度算法存在的不足,相关学者在 Kubernetes 调度模块的基础上,引入了一系列新的方法. 其中,Guerrero 等[5]针对 Kubernetes调度算法只关注物理资源使用和一般阈值的问题,提出了一种基于遗传算法的优化策略,使用非支配排序遗传算法-Ⅱ(non-dominated sorting genetic algorithm-Ⅱ)来优化容器分配和弹性管理,实现了系统配置、系统性能、系统故障和网络开销四个维度的优化目标;Liu 等[6]提出了一种面向大数据应用的容器调度改进算法——基于 Kubernetes 的粒子群算法,实现了集群性能以及资源利用率的提高;Wei 等[7]针对Kubernetes调度算法只关注当前最优节点,不考虑资源使用成本及能耗开销的问题,提出了一种结合蚁群算法和粒子群优化算法的调度模型,相比原算法,实现了能耗成本的降低和节点最大负载的下降;Mao 等[8]提出了一种 SpeCon 调度策略,这是一种针对短期深度学习应用程序进行优化的新型容器调度程序,在单个工作的完成时间上提升明显;Fu 等[9]同时衡量集群的即时资源利用率和对未来资源利用率的估计,提出了一种基于进度的容器调度方案——ProCon,它平衡了集群间的资源竞争,减少了任务的完成时间;El Haj Ahmed G 等[10]为 Kubernetes 构建了一个动态调度平台——KubCG,它的调度程序通过考虑 Pod 的时间线和有关容器执行的历史信息来优化新容器的部署,并将完成不同任务的时间减少至原始时间的 64%.

同时,目前针对 Kubernetes 调度算法的研究依然存在一些不足. 首先，现有方法虽然在一定程度上解决了 Spread 算法指标单一的问题,引入了诸如网络带宽、磁盘 IO 等因素,但未能考虑到容器间通信对于集群性能以及能耗的影响. 其次，现有方法在对 Pod 调度时,将其视为简单的线性结构,忽视了 Pod 之间的关联性. 而在真实的生产环境中,Pod 之间的结构更倾向于二维结构，保留 Pod 之间的关联性进行调度,对于集群整体性能的稳定以及主机功耗的降低是至关重要的.

针对目前调度策略存在的问题,本文提出了一种基于 Erdős-Rényi 模型[11]的 Pod 组合调度策略. 按照概率将 Pod 的调度问题转换为最优组合问题，并通过 Cloudsim 平台以及 Planetlab 数据集进行仿真模拟,实现了对时间效率以及能耗两个指标的优化.

1 相关背景

1.1 Kubernetes 编排系统

Kubernetes 是 Google 推出的一款基于 Golang 开发的可移植、可扩展的工具,用于管理容器化的工作负载和服务. 它消除了容器化应用程序在部署、伸缩时涉及到的许多手动操作,Kubernetes 可以简单高效地管理容器集群,构成这些集群的主机还可以跨越公有云、私有云以及混合云[3]. Kubernetes 系统架构如图 1 所示,遵循客户端/服务端(C/S)架构,系统架构分为 Master 和 Node 两部分. Master 作为服务端,Node 作为客户端. Kubernetes 系统具有多个 Master 服务端，服务端和客户端都由许多个组件协同完成工作[12],具备较强的稳定性.

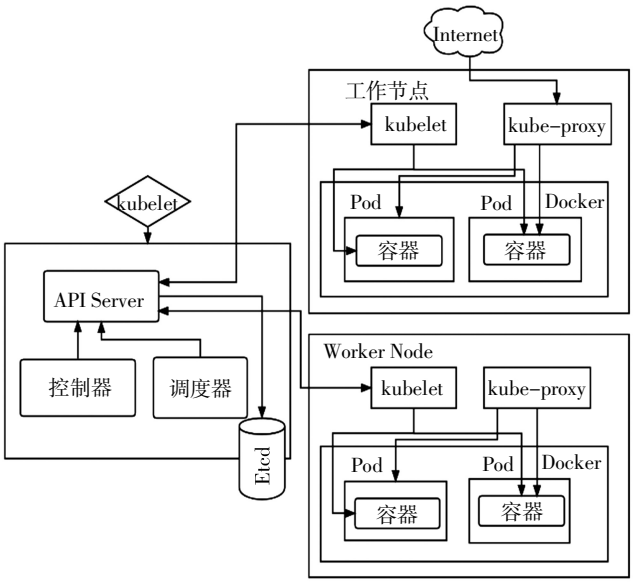
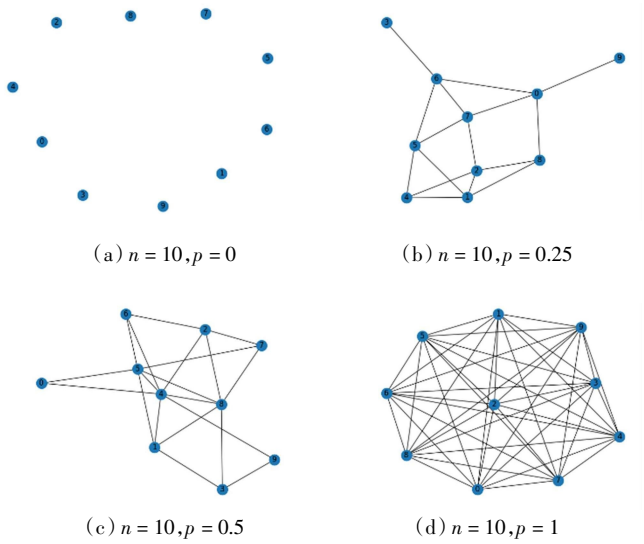


图 1　Kubernetes 架构

1.2　Erdős–Rényi 模型

在图论的数学领域,Erdős–Rényi 模型是用于生成随机图或随机网络演化的两个密切相关的模型之一[13]. 其以匈牙利数学家 Paul Erdős 和 Alfréd Rényi 的名字命名,并于 1959 年首次提出[11]. 而 Edgar Gilbert 在此基础上引入了另一个模型[14]. 在 Gilbert 引入的 Erdős–Rényi–Gilbert 模型中，每条边都有固定的存在或不存在的概率,与其他边无关. 本文使用到的模型定义为:给定 n 和 p,$G(n,p)$的定义是有个顶点,并且两个顶点之间以概率 $p \in [0,1]$来决定是否连边.

在 $G(n,p)$模型中,根据顶点数量 n 和连边概率 p 的不同会随机生成不同的简单图[15],如图 2 所示.



(a) $n=10, p=0$　(b) $n=10, p=0.25$　(c) $n=10, p=0.5$　(d) $n=10, p=1$

图 2　随机图的不同形式

在一个 $G(n,p)$模型中,它最终会生成一个图集合,其中节点的度的平均值为

$$<k> = \sum_{m=0}^{1} \frac{2m}{n} P(m) = (n-1)p \tag{1}$$

式中的度的平均值是一个常数,表示为 c.

此外,在 $G(n,p)$模型中,图中任意一个顶点的度为 k 时的概率为

$$P_k = \begin{pmatrix} n-1 \\ k \end{pmatrix} p^k (1-p)^{n-1-k} \tag{2}$$

同时,当 $n \to \infty$ 或 $n \gg k$ 时,图的度分布呈现出泊松分布的性质,当固定 $c = 15$ 时,分别取定点个数为 100,1 000, 10 000,100 000 时， 模型的度分布如图 3 所示.

根据 Erdős 和 Rényi 的相关研究[15],此时模型的度分布满足

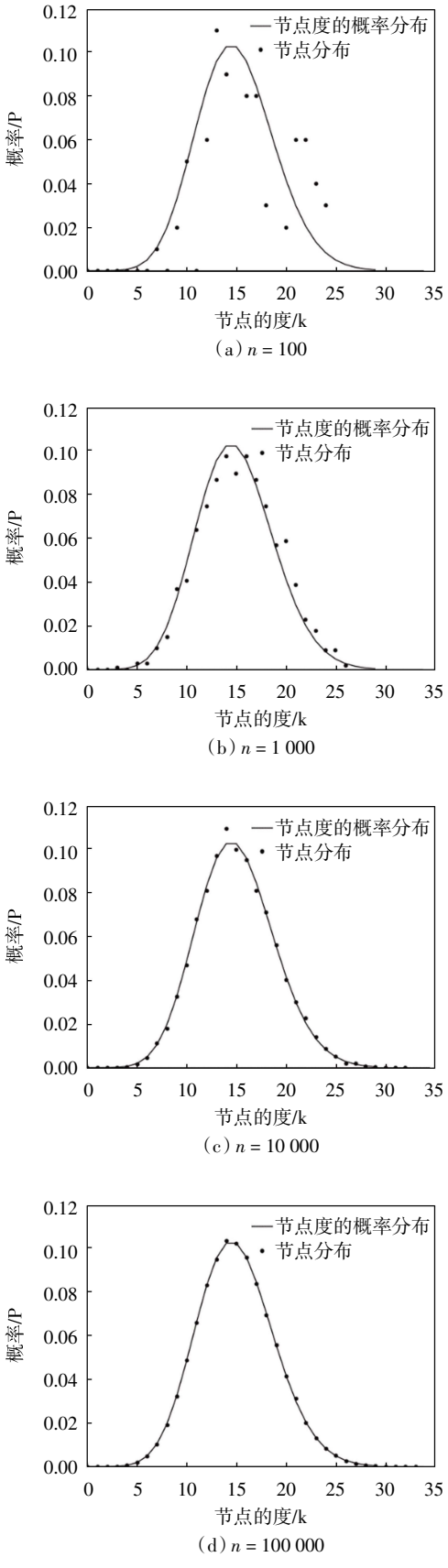


图 3　随机图的度分布

$$P_k = \frac{(n-1)}{k!} p^k e^{-c} = e^{-c} \frac{c^k}{k!} \tag{3}$$

2 调度策略

2.1 模型原理

在 Kubernetes 集群内部,Pod 之间构成的网络类似于图结构,每一个 Pod 即为图中的一个节点,而连接两个节点的加权连线表示为这两个节点之间的关联度,本文使用通信频率这一指标来衡量其关联程度. 因此,可以将集群中 Pod 的通信抽象为无向加权图来描述.

假设一个集群中存在 4 个工作节点,每个节点内部运行着 4 个 Pod, 那么这整个集群中存在 16 个 Pod,如图 4 所示.

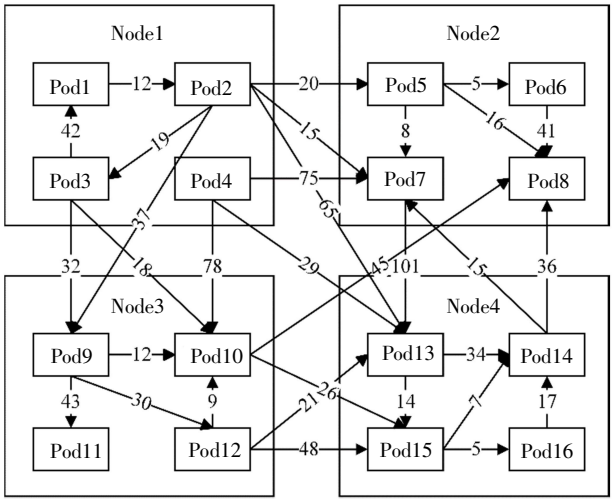


图 4 Kubernetes 集群通信模拟

采用的权值指标为通信频率,以此来衡量 Pod 的关联程度. 只考虑其通信频率的情况下, 可以无需考虑方向性,因为通信频率并不会随着两个 Pod 通信方向的改变而改变. 可以根据该权值分布将 Pod 网络抽象为二维图结构,创建该图结构的算法伪代码如下

Algorithm 1 Create Graph

Input:*podnum*,*arcnum*

1: for (i = 0;i<*podnum*;i++) do

Input: *POD*[i]. data

2: Initinlization *POD*[i].*firstarc* = NULL

3: end for

4: for (k = 0;k<*arcnum*;++k) do

Input: Connects two vertices of an edge $v1$,$v2$

5:i = LocatePod(G,$v1$);=LocatePod(G,$v2$)

6:$p1$=new ArcNode

7:$p1$. adjvex=j

8:$p1$. nextarc=*POD*[i].firstarc

9:$p2$ = new ArcNode

10:$p2$. adjvex=i

11:$p2$. nextarc=*POD*[i].firstarc

12: end for

Output: Graph G

在 Algorithm 1 中, 首先初始化集群中 Pod 数量 podnum,同时根据 Pod 之间的通信情况生成连线数量 arcnum;然后依次遍历每个 Pod 节点,将该 Pod 的权值等信息存入领接表中,并遍历每条 Pod 间的连线,将该连线所依附的两个 Pod 以及连线权值插入图中;最后返回生成的图结构如图 5 所示. 该算法时间复杂度和空间复杂度均为 O(podnum + arcnum).

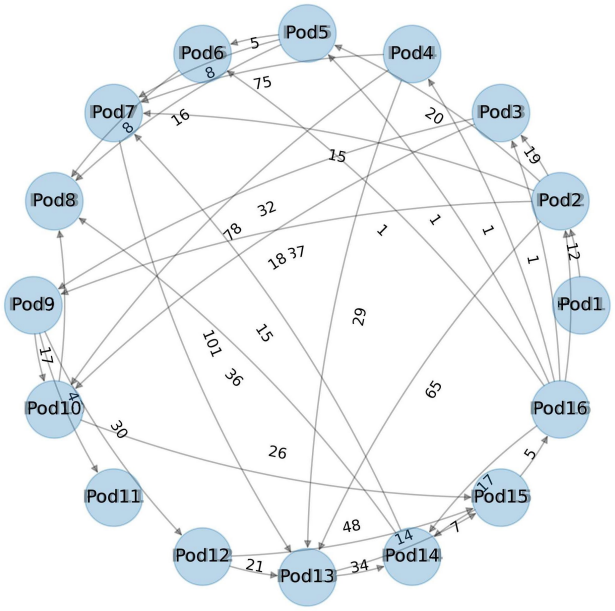


图 5 容器网络的图结构

Erdős–Rényi 模型与渗流理论关联紧密[16],同时,作为图结构,该模型也能很好地被应用到容器网络的研究中. 需要注意的是,针对不同的场景,需要对模型进行适当地调整,以满足不同的使用环境.

可以先假设集群中存 n 个 Pod,记为

$$N_i \in \{N_1, N_2, \cdots, N_n\} \tag{4}$$

两个互相连接的 Pod 之间的权值记为

$$i_{\langle m,n\rangle} \in \{i_{\langle 1,2\rangle}, i_{\langle 1,3\rangle}, \cdots, i_{\langle n-1,n\rangle}\} \tag{5}$$

该权值即表示两个 Pod 之间的通信频率,因此可以得到领接表,如表 1 所示.

基于以上权值的分布情况,可以构造出一张无向加权图,假设其为原始图 G. 对于原始图 G,其度是确定的,可以表示为 k.

同时,根据式(2),该图的度的平均值为

$$c = (n-1)p \tag{6}$$

表 1　权值分布

Pod 编号	1	2	3	…	n
1		$i_{\langle 1,2\rangle}$	$i_{\langle 1,3\rangle}$	…	$i_{\langle 1,n\rangle}$
2	$i_{\langle 2,1\rangle}$		$i_{\langle 2,3\rangle}$	…	$i_{\langle 2,n\rangle}$
3	$i_{\langle 3,1\rangle}$	$i_{\langle 3,2\rangle}$		…	$i_{\langle 3,n\rangle}$
…	…	…	…	…	…
$n-1$	$i_{\langle n-1,1\rangle}$	$i_{\langle n-1,2\rangle}$	$i_{\langle n-1,3\rangle}$	…	$i_{\langle n-1,n\rangle}$
n	$i_{\langle n,1\rangle}$	$i_{\langle n,2\rangle}$	$i_{\langle n,3\rangle}$	…	

因此,根据式(3),可以得到该模型的阈值概率满足以下关系

$$P=\frac{[(n-1)P]^k}{e^{(n-1)p}k!} \tag{7}$$

当 $n\gg k$ 时,可以得到阈值概率 P 的计算公式为

$$P=\frac{1}{n-1}\ln[\frac{(n-1)^k}{k!}] \tag{8}$$

式中:k、n 均为确定的常数值,因此,对于原始图 G,可以得到该模型任意两个节点之间连边的阈值概率 p. 它是一个关于节点数量和图中节点的度的函数式.

同时,在给定的原始图 G中,对任意一个节点 i,都需要根据其权值大小,评估其在该模型框架下两点之间存在连线的概率,以此来表示在图 G中,其已连接的边的影响程度. 因为该节点 i 在原图中的度是确定的,表示为 k_i. 同时,所有与 i 相连的节点记为集合 N_i.

根据式(2)中单个节点的度的概率计算公式以及综合权值,得到节点与节点连边的概率为

$$P_{i,j}=\begin{pmatrix} n-1 \\ k_i \end{pmatrix}P(1-P)^{n-1-k}\frac{\langle i,j\rangle}{\sum\langle i,k\rangle} \tag{9}$$

其中:$k\in N_i$ 且 $k\neq i$.

式 9 表示在原始图 G 中, 节点 i 与节点 j 之间连线的概率,也可以用来刻画其连线对于整个图结构的影响程度,根据其影响程度大小,进行保留或者“裁剪”操作. 具体为:

(1) 若 $P_{i,j}>P$,表明该连线概率大于阈值概率,影响程度较大,应进行保留操作;

(2) 若 $P_{i,j}<P$, 表明该连线概率小于阈值概率,影响程度较小,应进行“裁剪”操作.

通过以上操作,可以擦除掉原始图中一些影响较小的连线,保留下影响程度较大的连线,从而将整个图分解为若干个大小不一的连通分量. 每一个连通分量,则代表在 Kubernetes 集群中,他们之间的通信更为频繁,对于整个系统的效率、性能以及能耗影响更大. 因此, 需要将这些 Pod 调度到同一个物理或虚拟节点上,以次来保证集群性能.

2.2　调度流程

本文所提出的调度策略流程如图 6 所示.在Kubernetes原生的调度机制上引入监测模块和分析模块,监控模块通过在 Istio[17]中,利用 ServiceMesh[18]中的 SideCar 模式增加监测模块, 用以收集集群中 Pod 间的通信指标,包括通信响应时间、跨节点通信频率等. 然后将这些数据发送到分析模块, 分析模块收到这些数据后,首先会将这些数据以及 Pod 之间的关联性进行抽象建模,将其转换为无向加权图结构;然后使用改进的 Erdős–Rényi 模型对图结构中的连线按照概率(关联程度)进行保留或删除操作,直至将原来的连通图拆解为若干大小不一的联通子图;最后将分解的结果发送至 Informer,将其重新放入调度队列,由调度器实现最后的调度工作.

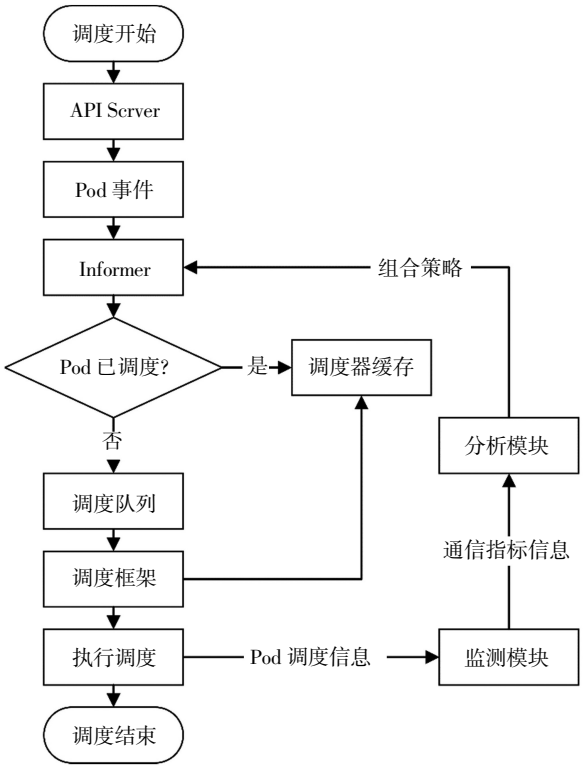


图 6　调度流程

该调度策略具体的调度流程如下.

步骤一:初始化 n 个新 Pod,并分配至不同的节点.

步骤二:Pod 调度信息发送至监测模块,同时该模块开始收集 Pod 之间通信频率指标.

步骤三: 监测模块将收集到的数据发送至分析模块.

步骤四:分析模块生成 Pod 间的组合模型,并发送至 Informer.

步骤五:Informer 根据分析模块计算的组合模型,将其重新加入调度队列.

步骤六:调度器将 Pod 组调度至最佳节点.

3 实验结果与分析

本章采用 CloudSim 平台[19]作为仿真平台,数据集为 PlanetLab 数据集[20],以此进行实验仿真.并与 Kubernetes 中采用的 Spread 调度算法进行对比,以验证所提出算法的有效性.

3.1 实验环境

本次实验的软件环境为:Windows10 64 位、JDK 1.8.0_301、IDEA 2021.2 、CloudSim 5.0.硬件环境为:Intel i5-11600KF、NVIDIA GeForce RTX 3060 Ti、16 GB.

3.2 参数选择与仿真结果

本文采用 CloudSim 5.0 仿真平台中的 container-vm-pm 架构方案,即将容器放置在虚拟机上、虚拟机放置在物理机节点之上.实验采用了该仿真平台集成的共享策略分配 CPU 资源,即同一个 CPU 在相同的时间之内可为多个不同的容器提供计算资源.

此外,在进行仿真时,首先需要实例化一个云计算平台.云计算平台包括一个或多个数据中心,每个数据中心包含若干物理服务器、虚拟机和容器.通过统计云任务的完成时间来评估集群的整体性能,并和 Kubernetes 中的 Spread 调度算法进行对比.实验结果表明,本文提出的调度策略可以有效降低云任务的时间开销及主机能耗.

为了模拟更加真实的 Kubernetes 集群环境,减小资源配额对于实验结果的影响,对虚拟机、物理机、容器的各项资源配置均采用层级递进的配置方案.实验过程中具体的参数设置如表 2、表 3、表 4 所示.

表 2 虚拟机参数

虚拟机编号	计算资源/Mips	内存/Mb	带宽/Mbps
1	32 669	1 024	500
2	56 840	1 024	600
3	78 905	2 048	700
4	95 274	2 048	800
5	153 075	4 096	1 000

表 3 物理机参数

物理机编号	计算资源/Mips	内存/Mb	带宽/Mbps
1	244 035	65 536	10 000
2	325 062	131 072	10 000
3	401 875	262 144	10 000

表 4 容器参数

容器编号	计算资源/Mips	内存/Mb
1	3 036	128
2	7 208	256
3	14 063	512

通过 10 次实验,每次实验初始化一组云任务,将这些云任务分别通过 Spread 调度算法与本文提出的基于 Erdős-Rényi 模型的调度策略分别进行放置.设置容器数量 n 为 100,度的平均值 k 为 5,同时不断提高任务长度,并获取任务的完成时间,因此可以得到 10 组任务完成的时间开销数据,如表 5 所示.通过分析实验数据得出,在云任务时间开销对比上,改进的调度策略均优于 Spread 算法,对比效果如图 7 所示.同时,整体来看,降低的时间开销与总任务长度大致成正比关系,如图 8 所示.即任务长度越大,下降的时间消耗也多.从下降率来看,整体上的时间开销,相较于 Kubernetes 的 Spread 调度算法,平均下降了 22%.

同时,分别将容器数量 n 设置为至 10,25,50,75,100,以此验证集群规模对于算法性能以及主机能耗的影响.实验结果分别如图 9、图 10 所示,根据实验结果可以看出,在不同规模的集群下,本文提出的容器组合调度策略在云任务的完成时间上均优于 Spread

表 5 云任务执行时间对比 ms

实验编号	任务长度	Spread 算法	ER 模型
1	3 000 000	702.149 899 8	550.117 79
2	4 000 000	935.866 618 7	733.157 075 9
3	5 000 000	1 169.583 29	916.196 381 1
4	6 000 000	1 403.299 987	1 099.235 668
5	7 000 000	1 637.016 611	1 282.274 946
6	8 000 000	1 870.733 282	1 465.314 25
7	9 000 000	2 104.449 995	1 648.353 538
8	1 000 000	2 338.166 654	1 831.392 82
9	2 000 000	4 675.333 453	3 661.785 736
10	3 000 000	7 012.500 216	5 492.178 626

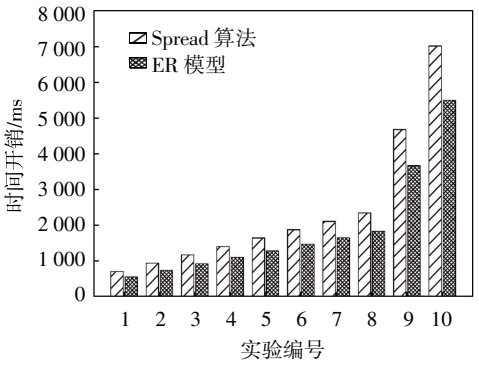


图 7 时间开销对比

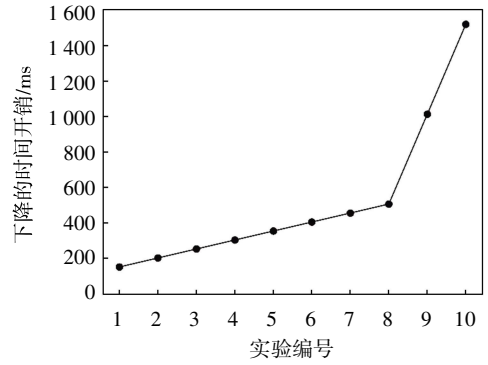


图 8　下降的时间开销

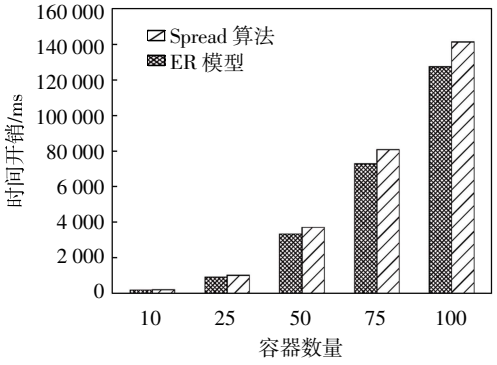


图 9　时间开销对比

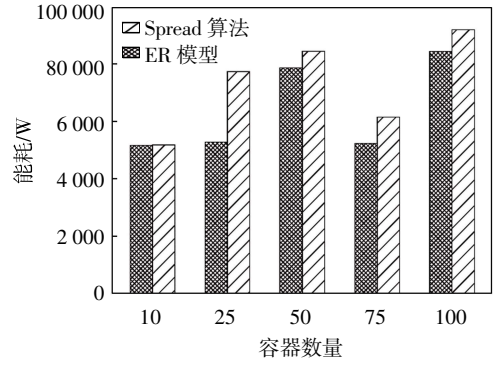


图 10　能耗对比

算法,算法性能整体上随着容器规模的扩大而更加显著. 同时能耗方面相较于 Spread 算法也更优，降低了大约 5%~30%.

4　结　语

本文提出了一种基于改进的 Erdős–Rényi 模型的容器组合调度策略,通过将关联性较大的一组 Pod 调度至同一个物理或者虚拟节点之上来降低集群的性能损耗与能耗开销. 通过实验对比分析，本文提出的方法调度策略可以在一定程度上提升集群的整体性能,降低能耗. 同时,该模型在得到一定成果的同时依然存在一些不足之处,需要在今后的工作中不断完善. 主要如下:①将模型的权值进行改进,以便使其可以用于容器的初始调度问题;②将监测模块和分析模块使用 Golang 语言进行代码实现,并将其与微服务相结合,在真实的 Kubernetes 集群中加以实现,以获取真实环境下的更多容器数据,同时引入更多指标,诸如亲和度、资源利用率等,使其能在多维方向上对集群多个指标进行优化.

参考文献:

[1] ZHANG S,ZHANG S,CHEN X,et al. Cloud computing research and development trend[C]// 2010 Second International Conference on Future Networks. NewYork: IEEE,2010: 93–97.

[2] 武志学. 云计算虚拟化技术的发展与趋势[J]. 计算机应用,2017,37(4): 915–923.

[3] LUKSA M. Kubernetes in action[M]. 北京: 电子工业出版社,2017.

[4] 吕元琛. 容器云环境下容器调度策略的研究与实现[D]. 大连: 大连理工大学,2020.

[5] GUERRERO C,LERA I,JUIZ C. Genetic algorithm for multi–objective optimization of container allocation in cloud architecture[J]. Journal of Grid Computing,2018,16(1): 113–135.

[6] LIU B,LI J,LIN W,et al. K–PSO: an improved PSO–based container scheduling algorithm for big data applications [J]. International Journal of Network Management,2021,31(2): 62–67.

[7] WEI G Z,XI L M,JIN Z Z. Research on kubernetes´ resource scheduling scheme[C]// Proceedings of the 8th International Conference on Communication and Network Security. New York: Association for Computing Machinery,2018: 144–148.

[8] MAO Y,FU Y,ZHENG W,et al. SpeCon:speculative container scheduling for deep learning applications in a kubernetes cluster[J]. IEEE Transactions on Services Computing,2020,14: 16–28.

[9] FU Y,ZHANG S,TERRERO J,et al. Progress–based container scheduling for short–lived applications in a kubernetes cluster [C]// 2019 IEEE International Conference on Big Data (Big Data). NewYork: IEEE,2019: 278–287.

[10] EL H A G,GIL C F,COSTA M E. KubCG: a dynamic kubernetes scheduler for heterogeneous clusters[J]. Software: Practice and Experience,2021,51(2): 213–234.

[11] 龚　正. Kubernetes 权威指南: 从 Docker 到 Kubernetes 实践全接触[M]. 北京: 电子工业出版社,2016.

[12] LEWIS T G. Network science: theory and applications [M]. Amsterdam: Elsevier Press,2011.

(下转第 227 页)