2023 年 3 月第 50 卷第 3 期
doi:10. 3969/j. issn. 1001-5922. 2023. 03. 045 Vol.50 No.03,Mar.2023

基于负载均衡的Kubernetes改进优化算法研究

梁兆楷[1],高建明[2]

(1. 广东电网有限责任公司 广州供电局,广东 广州 510000;
2. 南方电网数字电网科技(广东)有限公司,广东 广州 510000)

摘 要: 为了改善Kubernetes资源调度算法的负载均衡性,针对传统的Kubernetes默认调度算法只考虑了节点的CPU利用率和内存利用率,未考虑磁盘IO利用率和网络带宽利用率的问题,提出一种基于鸽群算法的Kubernetes资源调度算法。选择CPU、内存、磁盘IO和网络带宽平均资源利用率标准差作为适应度函数,通过鸽群算法的调度分配实现资源的最佳分配。研究结果表明,PIO-Kubernetes算法提高了集群负载能力,并且具有更强的并行处理能力。

关键词: 鸽群算法;负载均衡;Kubernetes;平均资源利用率
中图分类号: TP391.9 **文献标志码:** A **文章编号:** 1001-5922(2023)03-0192-05

Study on Kubernetes improved optimization algorithm based on load balancing

LIANG Zhaokai[1], GAO Jianming[2]

(1. Guangzhou Power Supply Bureau of Guangzhou Power Grid Co., Ltd., Guangzhou 510000, Chian;
2. China Southern Digital Power Grid Technology (Guangdong) Co., Ltd., Guangzhou 510000, Chian)

Abstract: The default scheduling mechanism of Kubernetes only considers the resource utilization of CPU and memory of a single node, but does not consider the resource utilization of disk IO and network bandwidth. In order to balance the load of the whole cluster, a Kubernetes resource scheduling algorithm based on pigeon flock algorithm is proposed. The standard deviation of average resource utilization of CPU, memory, disk IO and network bandwidth was selected as the fitness function, and the optimal resource allocation was achieved through the scheduling and allocation of pigeon swarm algorithm. The results show that the PIO-Kubernetes algorithm improves the cluster load capacity and has stronger parallel processing ability.

Keywords: pigeon-inspired optimization algorithm; load balancing; kubernetes; average resource utilization

随着我国用电量持续增加,我国电力系统始终处于高负荷工作状况,其主要原因为电力系统无法有效承受集群负载能力,易使电脑系统崩溃,且长期的高负荷工作也会使电脑CPU运行温度高,导致线路老化,硅脂材料开裂。且硅脂开裂易使传统导热填料以金属粉体、氧化物、碳化物、氮化物等为主的材料发生变化。伴随着随着互联网的快速普及以及网络应用规模的不断扩大,服务器产生的数据量呈现

收稿日期:2022-11-16;修回日期:2023-03-06
作者简介:梁兆楷(1993-),男,本科,助理工程师,研究方向:数据库、网络安全;E-mail:liangzhjia@126.com。
引文格式:梁兆楷,高建明. 基于负载均衡的Kubernetes改进优化算法研究[J]. 粘接,2023,50(3):192-196.

指数级增长，使得云计算技术得到了快速发展和应用。云计算是分布式计算的一种，通过网络“云”将丰富的网络数据资源进行统一分析、处理、计算、管理和调度，可以极大提高电脑系统运行效率，减少电脑内部材料老化及损伤，且由于长时间受到高温影响破坏，从而导致防护层的损坏，进而导致绝缘材料性能下降。在云计算模式下，软件、硬件和平台等网络资源以服务的方式提供给使用者，有效解决用户基础设施建设和系统维护成本高、效率低下以及网络资源的利用不均的问题[1]。传统云计算架构中基础设施即服务（IaaS）在虚拟机上作为基础单元来安排资源分配，但是，虚拟机的计划是一种粗粒度的资源分配方式，其运行速度较慢[2-3]。随着 Docker容器技术的快速发展，以容器为基础的虚拟化技术已经逐渐被云计算开发者和云服务提供商所青睐。容器技术相对于虚拟机来说节省了资源、启动速度快、镜像小以及应用部署灵活等优点[4]，可以降低电力系统负载能力，减少系统硬件元器损伤，同时，电荷的聚集、涡流、绝缘介质的损耗，都会产生额外的热量，导致系统线路的温度上升。长时间的超负荷运转，高温会使绝缘加速老化，最终导致绝缘破裂。然而，实际工作中需要管理的容器数量较为庞大且容器之间的关系相当复杂，一般会使用某个工具对数量庞大的容器进行编排、管理和控制，其中以Kubernetes[5]为代表的容器编排工具是当前较为热门的编排管理工具。

为了改善Kubernetes资源调度算法的负载均衡性，提高电力系统处理信息能力，针对传统的Kubernetes默认调度算法只考虑了节点的CPU利用率和内存利用率，未考虑磁盘IO利用率和网络带宽利用率的问题，提出一种基于鸽群算法（PIO）的Kubernetes资源调度算法。选择鸽群算法作为Kubernetes资源调度算法，在计算适应度阶段，选择CPU、内存、磁盘IO和网络带宽的平均资源利用率标准差作为适应度函数，通过鸽群算法的调度分配实现资源的最佳分配，进一步降低系统过载对元器件线路、硅脂、内存条的影响。

1 Kubernetes Scheduler调度算法

Kubernetes资源调度的核心组件为Kubernetes Scheduler。Kubernetes Scheduler的主要功能是按照指定的调度算法将Controller Manager或API Server新建的待调度Pod绑定到集群中某个合适的工作节点，与此同时将待调度的Pod和工作节点的绑定信息写入ETCD组件。之后，Kubernetes调度器发出的待调度Pod和工作节点的绑定信息由工作节点通过Kubelet监测到，并从ETCD组件中读取待调度Pod配置文件完成容器应用的启动。

根据调度策略，Pod被Scheduler部署到不同的节点中，Pod的分配模型如图1所示。

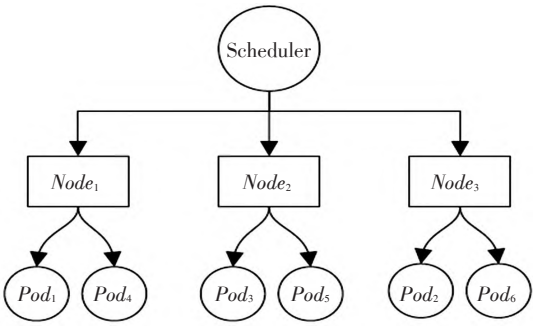

图1 Pod在Node的分配模型

Fig. 1 Pod distribution model in node

假定系统中所有节点的集合为 $N=\{Node_1, Node_2, Node_3, \cdots, Node_n\}$，$n$ 为节点总数，$Node_i$ 为第 i 个节点。系统中所有待调度Pod的集合为 $N=\{Pod_1, Pod_2, Pod_3, \cdots, Pod_m\}$，$m$ 为待调度Pod总数，Pod_j 为第 j 个待调度Pod。假设在某个工作节点 $Node_i$ 上有一组Pod，工作节点中待调度Pod的分配情况可以用C表示，待调度Pod的分配情况 $C_i=\{Pod_1, Pod_2, Pod_3, ..., Pod_k\}$，$k$ 为 $Node_i$ 上待调度Pod的数量。如图1所示，将被调度Pod的分布情况：$C_1=\{Pod_1, Pod_4\}$，$C_2=\{Pod_3, Pod_5\}$，$C_3=\{Pod_2, Pod_6\}$。

图1描述了待调度Pod在工作节点的分配情况，Kubernetes默认资源调度算法一般是将待调度Pod分配到性能最好的工作节点上运行，增加了性能高的工作节点的负载，造成了性能簇的负载不平衡，导致了整个簇的服务品质下降。通过对被调度Pod和工作节点的配置关系的分析，可以看出，Kubernetes调度问题实际就是将 m 个Pod分配到 n 个节点中的排列组合问题，该问题是一个NP-Hard问题。针对NP-Hard问题，启发式算法是解决该类问题的主要方法之一。鸽群算法[6]是模仿鸽子归巢行为而提出的群智能优化算法，与其他智能算法相比，该算法具有调整参数少、原理简单、鲁棒性更强以及计算更加简单的优点，目前被广泛地应用于机器学习、参数优化等领域。为了改善Kubernetes资源调度算法的负载均衡性，针对传统的Kubernetes默认调度算法只考虑了

节点的CPU利用率和内存利用率，未考虑磁盘IO利用率和网络带宽利用率的问题，最大限度地发挥每个节点的性能，选择鸽群算法作为Kubernetes资源调度算法，引入网络带宽、磁盘IO评估指数并给出各种权值，采用平均资源利用基准差异作为适应性函数，通过鸽群算法的调度分配实现资源的最佳分配。

2 鸽群算法(PIO)

当距离目标很遥远时，信鸽会借助地球磁场和路标来抵达目标。PIO算法中鸽子归巢包含了地图罗盘算子和路标算子。在远离目标地点时，通过地球磁场识别方位，在接近目标地点时使用本地标志。在PIO中，利用地球的地磁与太阳为基础建立了地图与指示器的数学模型，而标志算子的建立则是以地标为基础的。

2.1 指南针算子

地图模型是基于地磁场，第i只鸽子的速度和位置更新公式为[7]：

$$V_i(t)=V_i(t-1)\times e^{-Rt}+rand\times[X_g-X_i(t-1)] \quad (1)$$

$$X_i(t)=X_i(t-1)+V_i(t) \quad (2)$$

式中：R为指南针算子；X_i和V_i为第i只鸽子的速度和位置；$rand$为[0,1]的随机数；X_g为鸽子最好位置。

2.2 地标算子

地标算子模仿导航工具地标对鸽子的影响。当鸽群接近目的地时，会依靠临近地标进行导航。如果某只鸽子熟悉地标，则径直飞向目的地；反之，如果不熟悉地标并且远离目的，则该只鸽子会跟随熟悉地标的其他鸽子飞行到达目的地。首先将鸽子的适应度值排序，然后每次迭代对其种群数量减半，假设中心鸽子熟悉地形，可以直接飞向目的地，其他的鸽子都在中心鸽子的引领下，向着目的地前进。鸽群位置更新策略如下：

$$N_p(t)=\frac{N_p(t-1)}{2} \quad (3)$$

$$X_c(t)=\frac{\sum X_i(t)\times \text{fitness}(X_i(t))}{N_p\sum \text{fitness}(X_c(t)-X_i(t-1))} \quad (4)$$

$$X_i(t)=X_i(t-1)+rand\times(X_c(t)-X_i(t-1)) \quad (5)$$

式(3)~式(5)中：$N_p(t)$为第t代鸽子种群规模；$X_c(t)$为第t代所有鸽子的中心位置；fitness(x)为每只鸽子的质量。

3 基于PIO-Kubernetes的资源调度算法

传统的Kubernetes默认调度算法，只考虑了节点的CPU利用率和内存利用率，未考虑磁盘IO利用率和网络带宽利用率，无法均衡每个节点的负载。基于上述评估标准，本论文提出了提高硬盘IO使用率及网路频谱使用量的方法。

CPU在特定节点上的利用率$L(Cpu_i)$定义如式(6)所示[8]：

$$L(Cpu_i)=\frac{\sum_{j=1}^{N}CpuPod_{ij}}{PodNum_i}\times 100\% \quad (6)$$

式中：$CpuPod_{ij}$为第i个Node节点上第j个Pod的CPU使用量；$PodNum_i$为第i个Node节点上Pod的总数。同理，可以计算出该Node上内存的利用率$L(Mem_i)$、磁盘IO的利用率$L(Disk_i)$和网络的利用率$L(Net_i)$，进而可以计算出该Node上的资源平均利用率：

$$L(N_i)=\frac{L(Cpu_i)+L(Mem_i)+L(Disk_i)+L(Net_i)}{4} \quad (7)$$

在实际应用部署中，不同类型的节点对评价指标的侧重点不同，引入权重W表征不同评价指标对节点负载的影响[9]。

$$L(N_i)=\frac{W(Cpu_i)\times L(Cpu_i)+W(Mem_i)\times L(Mem_i)+W(Disk_i)\times L(Disk_i)+W(Net_i)\times L(Net_i)}{\text{Count}(W)} \quad (8)$$

式中：$W(Cpu_i)$、$W(Mem_i)$、$W(Disk_i)$和$W(Net_i)$分别为CPU、内存、磁盘IO和网络的权重。

为实现Kubernetes集群的负载平衡，采用平均可用资源利用率标准差异作为确定的条件[10-13]。

首先，根据不同的分配方式，对资源的平均使用率$L(Clu_k)$进行了计算：

$$L(Clu_k)=\frac{\sum_{i=1}^{N}L(N_i)}{N} \quad (9)$$

式中：N为符合要求的可行解数量。

其次，根据不同Node上的资源评价利用率$L(N_i)$和不同分配方案的资源平均使用率$L(Clu_k)$。

因此，PIO优化Kubernetes集群负载均衡的适应度函数为(平均资源使用率标准差最小)：

$$\text{fitness}(\sigma(Clu_k))=\sqrt{\frac{\sum_{i=1}^{N}(L(N_i)-L(Clu_k))^2}{N}} \quad (10)$$

基于PIO-Kubernetes的资源调度算法算法步骤如下：

Step1：算法初始化：设定种群规模N_p、最大迭代次数T_{max}、指南针因子R、指南针算子执行次数N_{c1max}、地标算子执行次数N_{c2max}。根据Pod分配模型，Kubernetes调度问题可以看成0~1选择问题，1表示Pod被分配在这个节点上；反之不分配给这个节点：

Step2：通过对每个鸽子的随机定位和速度进行了随机分析，发现并将其最优解进行了分析；

Step3：完成路标操作和罗盘操作。根据式(1)、(2)等式对鸽舍的定位及转速进行了修正，再将各鸽群的适配性进行对比，发现并纪录出最佳最佳方案；

Step4：如果重复操作的数量超过了指北针操作的最大数目，就会终止目前的循环，这时进行地标操作，或者跳到Step3；

Step5：按鸽子的健康值对它们进行分类，按照式(3)~式(5)进行转角操作，储存最好的位置，并保留最好的解决方案；

Step6：判断算法终止条件。若算法超过最大迭代次数T_{max}，则输出最佳位置，最佳位置对应Kubernetes的最佳资源调度分配方案；否则，跳转至Step5。

4 实验分析

为验证PIO-Kubernetes进行资源调度的有效性和可靠性，通过虚拟机的方式部署在宿主机上，实验采用Kubernetes1.17。宿主机操作系统为Windows10系统，内存为16 GB，CPU为i7-8750H。

为了验证PIO-Kubernetes算法的优越性，将PIO-Kubernetes与PSO-Kubernetes、GA-Kubernetes和DE-Kubernetes相比，通用参数设置[14-17]：种群规模10，最大迭代次数100；粒子群算法(PSO)学习因子c_1=c_2=2；PIO算法指南针因子R=0.5，指南针算子执行次数N_{c1max}=80，地标算子执行次数N_{c2max}=20；遗传算法(GA)交叉概率p_c=0.7，变异概率p_m=0.1；差分进化算法(DE)缩放因子CR=0.5，交叉概率p_c=0.7，不同算法寻优对比图如图2所示。由图2可知，PIO-Kubernetes具有更快的收敛速度和寻优能力。

PIO-Kubernetes算法的并行计算效果如表1。

为了说明PIO-Kubernetes算法的并行性能，采用人工方式将原来数据扩充1 000、2 000、4 000和8 000倍，不同集群节点数的加速比Speedup如图3所示。文中分别取集群节点数4、8、16、32。

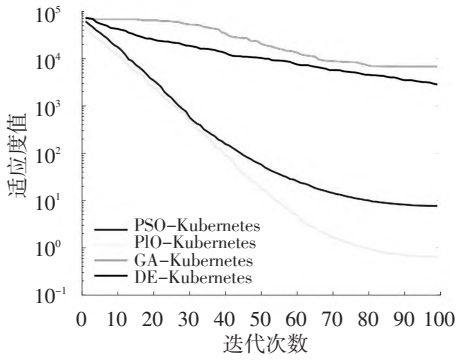

图2 不同算法寻优对比图

Fig. 2 Comparison of optimization by different algorithms

表1 并行性能

Tab. 1 Parallel performance

评价指标	1.5 MB	3.0 MB	6.0 MB	12.0 MB
计算时间/min	62	128	256	432
计算比	1.32	1.37	1.42	1.47
并行效率/%	66.64	68.97	71.61	74.11

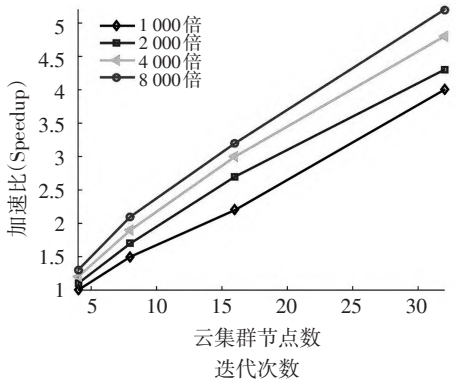

图3 加速比

Fig. 3 Acceleration ratio

由图3和表1可知，PIO-Kubernetes算法加速比随着数据集的增大而接近于线性，并且数据量越大加速效果越好；此外随着数据量的增大，加速比和并行效率也相应增加，说明PIO-Kubernetes算法适合分布式存储环境，具有良好的并行性能。

统计分析在120 s内3个节点CPU、内存、磁盘IO和网络带宽的使用率[18-20]，对比Kubernetes Schedule算法和PIO-Kubernetes算法在不同性能节点上CPU、内存、磁盘IO和网络带宽等使用率均值，不同资源使用率对比如表2所示。

由表2可知，采用PIO-Kubernetes算法后，工作节点的工作负荷会更大，如高性能内存工作节点$Node_1$使用Kubernetes Schedule算法和PIO-Kubernetes算法使用率分别为70.07%和78.50%，这是因为在高性能工作节点中，向其分配了较低的权重，减少

表2 资源使用率对比

Tab. 2 Comparison of resource usage

资源类型	节点	Kubernetes Schedule %	PIO－Kubernetes %
CPU	$Node_1$	62.79	73.23
	$Node_2$	50.67	37.86
	$Node_3$	38.16	37.52
内存	$Node_1$	70.07	78.50
	$Node_2$	39.61	35.32
	$Node_3$	39.78	34.97
网络带宽	$Node_1$	72.83	73.92
	$Node_2$	47.83	43.92
	$Node_3$	45.57	38.61
磁盘IO	$Node_1$	76.53	81.68
	$Node_2$	52.43	41.50
	$Node_3$	45.22	40.91

了工作节点的负荷，并向其分配了更多的Pod。但是低性能内存工作节点$Node_2$使用Kubernetes Schedule算法和PIO-Kubernetes算法使用率分别为39.61%和35.32%，表明采用PIO-Kubernetes方法可以有效地减少低工作结点的负荷。虽然高性能工作节点负载提高了，但因为工作结点具有良好的效能，所以没有给工作结点带来任何压力。因此，与Kubernetes Schedule方法相比较，PIO-Kubernetes方法能够有效地改善簇的性能。

5 结语

在Kubernetes系统中，对于Kubernetes的缺省排序方法，仅从CPU使用和存储效率两方面进行分析，以提高Kubernetes的负荷平衡。未考虑磁盘IO利用率和网络带宽利用率的问题，最大限度地发挥每个工作节点的性能，选择鸽群算法作为Kubernetes资源调度算法。在适配性阶段，引入网络带宽、磁盘IO评估指数和不同的权值，采用平均的资源利用基准差异作为适应性的函数，通过鸽群算法的调度分配实现资源的最佳分配。研究结果表明，PIO-Kubernetes算法提高了集群负载能力，并且具有更强的并行处理能力。然而，鸽群算法的初始种群是随机产生的，导致初始种群分布均匀性较差，使得优化时存在局部最优问题，后续将改进鸽群算法进行Kubernetes资源调度，提高模型的可靠性和精度。

【参考文献】

[1] 孔德瑾，姚晓玲. 面向5G边缘计算的Kubernetes资源调度策略[J]. 计算机工程，2021(2)：32-38.

[2] 徐正伦，杨鹤标. 基于Kubernetes调度器的服务质量优化调度算法研究[J]. 软件导刊，2018，17(11)：73-76

[3] 罗鹏，李景文. 基于PaaS环境的Kubernetes调度算法研究[J]. 中国新通信，2020，22(17)：43-45

[4] 林博，张惠民. 边缘计算环境下基于动态反馈的Kubernetes调度算法[J]. 信息技术与信息化，2019(10)：101-103

[5] 谭莉，陶宏才. 一种基于负载均衡的Kubernetes调度改进算法[J]. 成都信息工程学院学报，2019，34(3)：228-231.

[6] 顾清华，孟倩倩. 优化复杂函数的粒子群-鸽群混合优化算法[J]. 计算机工程与应用，2019，55(22)：4-52.

[7] 尚志刚，王力，李蒙蒙，等. 引入迷失探索与集群分裂机制的改进鸽群优化算法[J]. 郑州大学学报：工学版，2019(4)：25-31.

[8] 张文辉，王子辰. 基于组合权重TOPSIS的Kubernetes调度算法[J]. 计算机系统应用，2022，31(1)：195-203.

[9] 胡程鹏，薛涛. 基于遗传算法的Kubernetes资源调度算法[J]. 计算机系统应用，2021，30(9)：152-160

[10] 李华东，张学亮，王晓磊，等. Kubernetes集群中多节点合作博弈负载均衡策略[J]. 西安电子科技大学学报，2021，48(6)：16-22.

[11] 郝晓培，单杏花，李永，等. 基于Kubernetes的铁路客运营销深度学习平台的设计与实现[J]. 铁路计算机应用，2021，30(1)：57-61.

[12] 靳芳，龙娟. 一种面向Kubernetes集群的网络流量弹性管理方法[J]. 北京交通大学学报，2020，44(5)：77-86.

[13] 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.

[14] 刘刚，朱林，祁升龙，等. 基于拓扑结构变化的配电网自适应保护方法[J]. 电力科学与技术学报，2022，37(1)：106-112.

[15] 胡程平，陈婧，胡剑地，等. 配电台区线损率异常自动检测系统设计[J]. 粘接，2022，49(5)：188-192.

[16] 汪伟，于洋. 基于ASON的电力系统调度数据网业务体系设计[J]. 机械与电子. 2022，40(3)：17-20.

[17] 周石金，何晋，杨凡，等. IIDG接入对配电网保护的影响分析[J]. 现代电子技术. 2021，44(17)：151-156.

[18] 何昕雷. 基于模糊PID算法的自动控制研究[J]. 粘接，2022，49(3)：177-181.

[19] 董新涛，李文伟，李宝伟，等. 基于补偿零序压差原理的110 kV线路纵向故障保护方案[J]. 智慧电力，2022，50(9)：110-117.

[20] 谢珂，齐岩，李琛，等. 基于测量阻抗加速的反时限过流保护新方案研究[J]. 能源与节能，2022(2)：109-112.