计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(9):199−204 [doi: 10.15888/j.cnki.csa.006544]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Docker Swarm 集群的调度策略优化算法①

刘　梅[1,2], 高　岑[2], 田　月[2], 王　嵩[2], 刘　璐[2]

[1](中国科学院大学 计算机与控制学院, 北京 100049)
[2](中国科学院 沈阳计算技术研究所, 沈阳 110168)
通讯作者: 刘　梅, E-mail: 762369646@qq.com

摘　要: Swarm 是一种对集群中 Docker 镜像和容器进行管理的工具, 其在计算节点权值时可能会得到若干个相同权值的节点. 现有的 Swarm 调度策略只是将这些节点随机分配, 由于相同权值节点的资源负载情况并不相同, 所以将会造成节点负载不均衡. 针对上述问题, 本文提出一种动态调度算法对 Swarm 调度策略进行优化. 通过实验, 证明增加动态调度算法能够使集群中节点负载更加均衡, 同时提高集群的整体资源利用率.

关键词: 轻量级虚拟化; Docker; Swarm 集群; 负载均衡

引用格式: 刘梅,高岑,田月,王嵩,刘璐.基于 Docker Swarm 集群的调度策略优化算法.计算机系统应用,2018,27(9):199–204. http://www.c-s-a.org.cn/1003-3254/6544.html

Scheduling Strategy Optimization Algorithm Based on Docker Swarm Cluster

LIU Mei[1,2], GAO Cen[2], TIAN Yue[2], WANG Song[2], LIU Lu[2]

[1](School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: Swarm is a management tool for cluster Docker image and containers, and it may get a number of identical weights nodes in the calculation of weights. The existing Swarm scheduling strategy is just a random allocation of these nodes. Because the resources load of the same weight nodes are different, this will cause the load imbalance of the nodes. To solve this problem, this study proposes a dynamic scheduling algorithm to optimize the Swarm scheduling strategy. Through the experiment, it is proved that the dynamic scheduling algorithm can make the node load more balanced in the cluster, and improve the overall resource utilization of the cluster.

Key words: lightweight virtualization; Docker; Swarm cluster; load balancing

1　引言

Docker[1−3]是一个开源的引擎, 提供一种轻量级虚拟化的解决方案. 目前 Docker 已经获得操作系统 Linux 支持, 同时越来越多的大企业如谷歌, 微软, IBM 等开始青睐此项技术. 2014 年 12 月, Docker 公司发布了基于 Go 语言开发的原生态容器集群管理工具 Swarm, 主要用来对整个集群中的 Docker 镜像和容器进行管理, 而且以一个虚拟整体的方式将整个集群呈现给用户, Swarm 工具的发布促进了 Docker 在集群中的运用.

近年来, 随着云计算的迅速发展, 越来越多的公司利用 Docker 和 Swarm 工具在多个服务器上部署私有的 PaaS (平台即服务) 平台或 IaaS (基础设施即服务) 平台. Docker 虚拟机与 KVM (Kernel-based Virtual Machine) 一类的相比, 最明显的优势是创建速度和启停速度更快, 占用系统资源更少. 原生 Swarm 调度策略是在合适的节点上启动并运行 Docker 容器, 因此每个节点资源利用率的高低对整个集群的负载情况起决

① 收稿时间: 2018-01-30; 修改时间: 2018-02-27; 采用时间: 2018-03-08; csa 在线出版时间: 2018-08-16

© 中国科学院软件研究所 http://www.c-s-a.org.cn

定性作用. 目前 Swarm 内置三种调度策略 Random, Spread 和 Binpack.Random 策略是随机选择一个节点, 一般用于开发测试阶段. Spread 是优先选择权值小的节点 (该节点占用资源如 CPU, 内存最少), 以保证集群中所有节点资源的均匀使用. Binpack 是优先选择权值大的节点 (该节点占用资源如 CPU, 内存最多), 以保证更多空余节点. Swarm 在计算节点权值时可能会得到两个或多个相同权值的节点, 而调度策略只是将这些相同权值的节点随机分配, 并没有进行二次比较. 即使这些节点的权值相同, 但它们的 CPU 和内存使用情况也会不同. 所以可能会造成以下情况: ① 已使用较多 CPU 资源和较少内存资源的节点获得启动并运行容器的机会, 而该容器需求较多的 CPU 资源和较少的内存资源; ② 已使用较少 CPU 资源和较多内存资源的节点获得启动并运行容器的机会, 而该容器需求较少的 CPU 资源和较多的内存资源. 上述情况均会导致节点负载不平衡, 从而造成整个集群负载[4–7]不均衡, 降低集群的整体资源利用率.

针对上述问题, 本文在 Swarm 集群调度策略的基础上, 提出一种动态调度算法对 Swarm 调度[8–10]进行优化. 通过实验证明, 本算法能够使集群中节点负载更加均衡, 同时提高集群的整体资源利用率.

2 Docker Swarm 集群

2.1 关于 Docker

LXC (Linux Container)[11]容器技术是 Linux 系统中共享内核的操作系统级别的虚拟化技术, 通过虚拟容器和宿主机共享内核来提升容器的启停速度, 同时降低对内存等资源的消耗. 而 Docker 是基于 LXC 容器技术的一种实现, 一个 Docker 容器相当于一个虚拟机, 既可以拥有特定的操作系统, 也可以部署一个特定的应用, 同时独立于所运行的操作系统. 所以与传统的虚拟机相比, Docker 省去了部署应用时环境配置, 解决依赖等步骤, 并且没有任何中间层资源开销, 提升了资源利用率.

Docker 采用了客户端-服务器 (C/S) 架构, 架构如图 1 所示. 其中, Docker client 是客户端, Docker daemon 是一个后台模块, 用户通过 Docker-Client 发送请求, Docker daemon 对请求进行处理, 实现对镜像和容器的使用和管理. Docker daemon 模块主要包括 Docker server 和 Docker engine 两部分, Docker server 接收并调度分发 Docker client 发送的用户请求, 之后将具体的执行步骤交给 Docker engine, Docker engine 将每一个具体的操作抽象为一个类似操作系统中进程的 job 操作. Docker registry 是 Docker 的镜像存储服务端, 主要用于管理镜像的上传与下载. Docker driver 是驱动模块, 负责对 Docker 容器执行环境进行定制. Libcontainer 是 Docker 系统中使用 Go 语言实现的库, 其设计目标是希望不依赖其他库文件, 可以直接访问内核中与容器相关的 API.

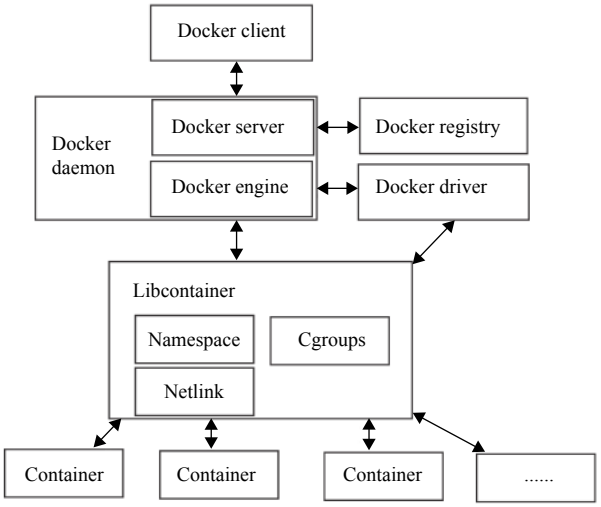

图 1 Docker 体系架构

2.2 Swarm 集群架构

Docker 公司在 2014 年 12 月初发布了 Docker 集群管理工具 Swarm, 用来统一管理由多个部署有 Docker 的物理机组成的集群中的镜像和容器.

Swarm 采用客户端-服务器 (C/S) 架构, 架构如图 2 所示.

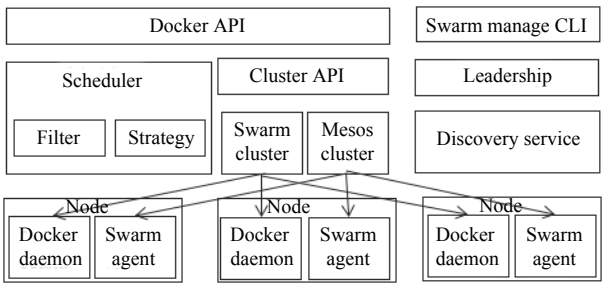

图 2 Swarm 架构

图 2 中, Docker API 负责管理镜像的生命周期. Swarm Manage CLI 负责对集群进行管理. LeaderShip

© 中国科学院软件研究所 http://www.c-s-a.org.cn

提供集群高可用机制 (High Avaliable), 防止单点故障. Cluster API 定义了集群接口, 主要用于对任务进行调度. Discovery service 是 Swarm 的发现服务, 主要用于在每个节点上注册一个 Agent 时, 将各个节点的 IP 端口上报, Manager 将从 Discovery service 读取各节点信息. Scheduler 是调度模块, 负责在容器调度时选择最优节点, 其主要分为两部分: Filter 和 Strategy. Filter 负责创建或运行容器时, 告诉调度器哪些节点是可用的, 它可以分为节点过滤和基于容器配置过滤, 用户在创建集群时, 需要根据节点功能分配给节点特定的标签, 一旦启动容器, Filter 会根据用户提供的标签, 过滤出匹配标签的节点集合供 Strategy 处理. Strategy 根据调度策略选择最优节点, Swarm 内置的调度策略包括 Random, Spread 和 Binpack 三种. 其中 Random 策略是随机选择一个节点, 一般用于开发测试阶段. Spread 是优先选择权值小的节点 (该节点占用资源如 CPU, 内存最少), 以保证集群中所有节点资源的均匀使用. Binpack 是优先选择权值大的节点 (该节点占用资源如 CPU, 内存最多), 以保证更多空余节点.

2.3 Swarm 集群的调度策略及分析

目前 Swarm 内置的调度策略有三种, 分别是 Random(随机), Spread (扩散) 和 Binpack (装箱), 集群默认的调度策略是 Spread. Random 调度是选择容器生成的节点位置是随机的, 一般用于开发测试阶段.

Spread 和 Binpack 调度首先根据申请容器的配置信息计算出 CPU 和内存利用率, 将其结果求和得到节点权重, 然后选择合适的节点运行容器. 其中, Spread 会优先选选择权值小的节点运行容器, 而 Binpack 会优先选择权值大的节点运行容器.

Random 策略的优点是实施简单, 但经常容易导致节点 CPU 过载和内存不足. 采用 Spread 策略, 虽然能够减少因节点故障而损坏容器的数量, 但是这种策略过多的占用了服务器资源 (例如内存资源). 而 Binpack 的优点是能将更多的容器运行在较少的节点上, 但是会造成节点负载过重.

Swarm 调度策略虽然简单有效, 但是其在计算节点权值时可能会得到两个或多个相同权值的节点, 而调度策略只是将这些相同权值的节点随机分配, 并没有进行二次比较. 即使这些节点的权值相同, 但它们的 CPU 和内存使用情况也会不同. 所以可能会造成以下情况: ① 已使用较多 CPU 资源和较少内存资源的节点获得启动并运行容器的机会, 而该容器需求较多的 CPU 资源和较少的内存资源; ② 已使用较少 CPU 资源和较多内存资源的节点获得启动并运行容器的机会, 而该容器需求较少的 CPU 资源和较多的内存资源. 上述情况均会导致节点负载不平衡, 从而造成整个集群负载不均衡, 降低集群的整体资源利用率.

3 基于 Docker Swarm 集群的动态调度策略

3.1 Swarm 集群的动态调度策略

根据上述问题, 本文提出了一种动态调度策略对 Swarm 调度策略进行优化. 通过动态收集节点信息, 实现对下一时刻节点资源使用情况的预测, 利用预测信息, 对权值相同的节点再次计算其权值, 进而再次排序, 优先选取权值小的节点启动和运行容器, 完成调度. 具体实现过程如下:

首先, 在每个节点上添加一个监控模块实时监控节点资源 (CPU 和内存资源) 的利用率, 输出为当前节点的 CPU 利用率和内存利用率. 其次, 在集群中添加一个信息收集模块, 用来收集在一段时间内监控模块传递过来的节点信息, 通过这些信息, 预测出下一时刻节点的资源使用情况. 最后, Scheduler 模块调用收集模块, 获取预测的节点信息, 其中, Scheduler 首先计算出节点 CPU 和内存利用率的的权值, 进行排序, 若存在相同权值的节点, Scheduler 会根据这些节点的预测信息进行第二次权值计算, 根据计算结果, 对它们进行第二次排序, 从而选择权值小的节点启动和运行容器, 完成调度功能.

3.2 Swarm 集群的动态调度算法的设计与实现

基于上述提出的动态调度策略, 本文提出动态调度的算法. 信息收集模块在获取最近时间段的节点资源使用信息后, 利用该算法, 实现对当前节点下一时刻 CPU 和内存使用情况的预测.

(1) 预测模型的选取

对 Swarm 集群中节点资源信息进行预测, 在选取预测算法时, 应考虑以下三点: 第一, 预测模型能够在一定误差内, 真实的反映出节点的 CPU 和内存资源使用情况; 第二, 对节点资源信息的预测都是实时预测, 所以应该保证算法具有较小的时间复杂度; 第三, 在预测时仅需要对最近时间段产生的若干个样本进行拟合. 鉴于上述考虑, 本文采用一元线性回归[12,13]模型对节点信息进行预测.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

在统计学中, 一元线性回归模型预测是回归分析与预测理论相结合的一种定量分析预测方法, 通过对预测对象和影响因素统计与分析, 找出它们之间的线性变化规律, 将变化规律用数学模型表达出来, 并利用数学模型进行预测. 其主要的优点包括实用性强, 预测结果精确和使用方法简便.

(2) 基于线性回归模型的节点负载预测

修改后的 Swarm 集群中的监控模块实时监控节点资源信息, 基于原生的 Swarm 调度策略只计算了 CPU 和内存两个维度的资源, 所以监控模块监控的节点信息包含 CPU 利用率和内存利用率, 假定节点的 CPU 利用率为 C, 内存利用率为 M. 首先, 模块获取前 t_n 时刻 $C=(<t_1,c_1>,<t_2,c_2>,<t_3,c_3>,\cdots,<t_n,c_n>)$, $M=(<t_1,m_1>,<t_2,m_2>,<t_3,m_3>,\cdots,<t_n,m_n>)$.

其次, 信息收集模块调用监控模块, 获取这些样本值, 采用一元线性回归分析方法进行模拟, 建立数学模型. 最后, 利用数学模型对 t_{n+1} 时刻的 c_{n+1} 和 m_{n+1} 进行预测. 具体实现过程如下:

步骤一. 设在 t_i 时刻监控对象的取值为 r_i, 时间序列 $T=<t_1,t_2,t_3,\cdots,t_n>$ 对应的监控对象取值为 $R=<r_1,r_2,r_3,\cdots,r_n>$. 假定存在 r 关于 t 的一次函数:

$$r(t)=at+b \tag{1}$$

其中, a 与 b 为不依赖 t 的常量.

每一个真实值与其预测值的随机误差 ε 之间相互独立, 且服从同一分布, 则:

$$\varepsilon=r-(at+b),\ \ \varepsilon\sim N(0,\sigma^2) \tag{2}$$

得出一元线性回归模型为:

$$r(t)=at+b+\varepsilon,\ \ \varepsilon\sim N(0,\sigma^2) \tag{3}$$

步骤二. 未知参数 a, b 的确立. 由每个样本 r 和监控值的随机误差 ε 之间相互独立可得出样本中 r 的联合密度函数为:

$$H=\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n e^{-\frac{\sum_{i=1}^{n}(r_i-at-b)^2}{2\sigma^2}} \tag{4}$$

设置 H 求最大值, 则得 a 与 b 的极大似然估计的值为:

$$\begin{cases}\hat{a}=\dfrac{\sum_{i=1}^{n}(t_i-\bar{t})(r_i-\bar{r})}{\sum_{i=1}^{n}(t_i-\bar{t})^2}\\ \hat{b}=\bar{r}-\hat{a}\bar{t}\end{cases} \tag{5}$$

式中, $\bar{t}$ 与 $\bar{r}$ 分别为样本中时间序列与监控值的平均值, 即满足:

$$\bar{t}=\frac{1}{n}\sum_{i=1}^{n}t_i \tag{6}$$

$$\bar{r}=\frac{1}{n}\sum_{i=1}^{n}r_i \tag{7}$$

得出 r 关于时间 t 的经验回归方程:

$$\hat{r}=\hat{a}t+\hat{b} \tag{8}$$

步骤三. 将进行预测的时间 t_{n+1} 代入式 (8), 得到预测值:

$$\hat{r}_{n+1}=\hat{a}t_{n+1}+\hat{b} \tag{9}$$

步骤四. 精度检测. 根据式 (10) 计算整个样本中监控对象的真实值和估测值之间的残差平方和.

$$L=\sum_{i=1}^{n}(r_i-\hat{r}_i)^2=\sum_{i=1}^{n}\left[r_i-\hat{a}t_i-\hat{b}\right]^2 \tag{10}$$

推导得出 $L/\sigma^2\sim\chi^2(n-2)$, 则整个样本的 σ^2 无偏估计量满足如下关系:

$$\hat{\sigma}^2=\frac{L}{n-2}=\frac{[S_{rr}-\hat{a}S_{tr}]}{n-2} \tag{11}$$

其中, S_{rr} 与 S_{tr} 满足:

$$S_{rr}=\sum_{i=1}^{n}r_i^2-\frac{1}{n}(\sum_{i=1}^{n}r_i)^2 \tag{12}$$

$$S_{tr}=\sum_{i=1}^{n}t_ir_i-\frac{1}{n}(\sum_{i=1}^{n}t_i)(\sum_{i=1}^{n}ri) \tag{13}$$

若样本的整体 $\hat{\sigma}^2$ 值满足 $\hat{\sigma}^2\leqslant 0.1$, 则认为预测模型的预测精度较高. 若样本的整体 $\hat{\sigma}^2$ 满足 $0.1<\hat{\sigma}^2\leqslant 0.2$, 则认为预测模型的预测精度一般. 若样本的整体 $\hat{\sigma}^2$ 满足 $\hat{\sigma}^2>0.2$, 则去掉样本首先出现的样本, 缩小样本集, 再利用剩下的样本去估计.

(3) 动态调度算法的运行过程

第一步, 分别获取当前节点的 CPU 利用率和内存利用率的时间序列 T 及其相对应的样本观测值 R.

第二步, 将 T 和 R 分别代入式 (6), 式 (7), 得到 $\bar{t}$ 和 $\bar{r}$, 再根据 H(式 (4)), 求出 $\hat{a}$ 和 $\hat{b}$, 确立经验回归方程.

第四步, 根据式 (11) 对预测模型进行精度检测, 评估预测模型是否符合要求, 若不符合要求, 缩小样本集, 返回第二步.

第五步, 对节点的 CPU 和内存的预测值进行权值计算. 计算权值的方法采用 Swarm 内置的.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4 实验结果与分析

本文是基于 Linux 操作系统进行实验的. 自 Docker 1.12 版本及该版本之后, Swarm 集成进了 Docker Engine. 本文选取的 Docker 版本为 Swarm17.03.0-ce. 搭建了 4 个 Swarm 集群, 集群 1, 2, 3 采用原生的 Swarm 搭建, 其中集群 1 的调度策略为 Random, 集群 2 的调度策略为 Spread, 集群 3 的调度策略为 Binpack; 集群 4 采用改进后的 Swarm 搭建. 对集群进行部署, 每个集群配置三个节点, 将运行容器的内存上限设置为 100 MB, 1 GB, 1 核的节点大约消耗 10% 的 CPU. 集群 1, 2, 3, 4 中各节点的性能为: 节点 1 为 3 GB, 4 核; 节点 2 为 2 GB, 2 核; 节点 3 为 1 GB, 1 核.

初始化集群中每个节点的资源利用状态, 设置节点整体的资源使用率为 S, 内存使用率为 M, CPU 使用率为 C, 则它们之间存在的的关系为: $S=M+C$; 每个节点资源的初始化状态如表 1 所示.

表 1 节点资源的初始状态 (单位: %)

节点索引	1	2	4
M	31	10	6
C	37	42	33
S	68	52	39

本实验通过模拟, 设定集群 1, 集群 2, 集群 3, 集群 4 中的节点 1 都是满负载, 剩余节点都为初始化状态, 然后分别创建等数量容器 40 个. 每个集群中的的各节点资源利用情况分别如表 2, 表 3 所示.

表 2 集群 1 和集群 2 中各节点资源利用情况 (单位: %)

集群	1 (Random 调度)			2 (Spread 调度)		
节点	1	2	3	1	2	3
M	75	44	23	82	33	11
C	45	62	25	39	62	48
S	120	106	48	121	95	59

表 3 集群 3 和集群 4 中各节点资源利用情况 (单位: %)

集群	3 (Binback 调度)			4 (动态调度)		
节点	1	2	3	1	2	3
M	90	42	6	39	40	35
C	65	31	33	46	52	50
S	155	73	39	85	92	85

由表 1 和表 2 可以看出, Swarm 内置的三种调度策略对每个节点的资源利用较不平衡. 将表 1 和表 2 中每个集群中的节点负载情况绘制成图 3 所示. 由图 3 的结果表明, 集群 1, 集群 2 和集群 3 分别利用 Random 策略, Spread 策略和 Binpack 策略完成调度时, 集群中的各个节点的负载相差较大, 每一个节点的资源利用情况很不均衡. 而集群 4 中增加一个动态调度算法后, 集群中每个节点的负载较为平衡, 这样就能够充分发挥每一个节点的性能, 进而提高集群的整体资源利用率, 同时, 集群整体的负载也更加均衡.

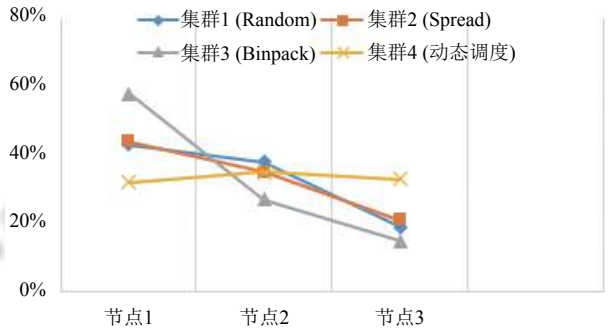

图 3 集群负载均衡对比图

5 结语

本文分析了 Swarm 集群框架及其内置的调度策略, 针对其调度策略在解决相同权值节点随机获取容器将会造成节点负载不均的问题, 提出一种动态调度算法来均衡集群的负载. 实验表明, 增加动态调度算法能够使集群中节点负载更加均衡, 同时提高集群的整体资源利用率.

参考文献

1 杨保华, 戴王剑, 曹亚仑. Docker 技术入门与实战. 北京: 机械工业出版社, 2015.
2 孙宏亮. Docker 源码分析. 北京: 机械工业出版社, 2015.
3 马越, 黄刚. 基于 Docker 的应用软件虚拟化研究. 软件, 2015, 36(3): 10–14. [doi: 10.3969/j.issn.1673-2022.2015.03.004]
4 Li X, Qian ZZ, Lu SL, *et al*. Energy efficient virtual machine placement algorithm with balanced and improve resource utilization in a data center. 2013 7th International Conference on Innovative Mobileand Internet Services in Ubiquitous Computing (IMIS). 2013. 313–319.
5 硕珺. PC 集群负载均衡调度策略研究[硕士学位论文]. 青岛: 中国石油大学, 2010.
6 王红斌. Web 服务器集群系统的自适应负载均衡调度策略研究[博士学位论文]. 长春: 吉林大学, 2013.
7 张丽梅. 基于负载均衡的云资源调度策略研究[硕士学位论文]. 银川: 宁夏大学, 2014.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

8 卢胜林, 倪明, 张翰博. 基于 Docker Swarm 集群的调度策略优化. 信息技术, 2016, (7): 147–151. [doi: 10.3969/j.issn.1671-3176.2016.07.041]

9 马晓光, 刘钊远. 一种适用于 Docker Swarm 集群的调度策略和算法. 计算机应用与软件, 2017, 34(5): 283–287. [doi: 10.3969/j.issn.1000-386x.2017.05.049]

10 李丹程, 王晓晨, 宋晓雪, 等. 基于 OpenStack 的资源负载预测方法研究. 计算机应用研究, 2014, 31(7): 2178–2182. [doi: 10.3969/j.issn.1001-3695.2014.07.063]

11 Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 2014, 1(3): 81–84. [doi: 10.1109/MCC.2014.51]

12 吴起, 宋立国, 张宇飞. 一元线性回归模型在负荷预测中的应用. 中国电力教育, 2010, (S1): 265–267.

13 张东华, 索传军. 基于线性回归法的网络信息资源评价模型的应用. 现代情报, 2007, 27(8): 10–12. [doi: 10.3969/j.issn.1008-0821.2007.08.003]

www.c-s-a.org.cn

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn