Software Engineering and Applications 软件工程与应用, 2025, 14(2), 346-358
Published Online April 2025 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2025.142031

Hans汉斯

基于改进人工鱼群算法的电商平台AR-ASFA分布式系统

杨赫宇，李洪军，范海平

上海理工大学光电信息与计算机工程学院，上海

收稿日期：2025年3月8日；录用日期：2025年4月15日；发布日期：2025年4月25日

摘　要

数字化、信息化和互联网的普及催生了电子商务的高速发展。面对急速膨胀的用户访问量，电商平台如何承载更多的用户流量并提高服务器响应性能等技术上仍存在瓶颈。本文从软硬件两方面进行系统优化，构建高性能分布式AR-AFSA系统。(1) AR (Application Router)架构配置三台JobManager服务器节点，分别接收三种方式的用户访问请求，运用流量分配机制分散系统的流量承载压力，并将用户请求按不同访问方式划分为四个流量队列进行调度。(2) 改进的人工鱼群算法(AFSA)进行容器调度，重新规划人工鱼的各行为执行顺序，增加对最优解寻找的可能性并加快局部收敛速度。(3) 设计人工鱼的参数与评价指标，为用户请求匹配足够资源的容器同时保证资源节约和系统负载均衡。最后在淘宝用户行为数据集以及多组对照实验下进行验证，AR系统可承载传统服务器三倍的流量压力。改进的人工鱼群算法相比对照算法可收敛至更优解，并且在服务器资源规模更为复杂的情况下，展示出更大的优势。

关键词

分布式系统，元启发式算法，电子商务

AR-ASFA Distributed System for E-Commerce Platform Based on Improved Artificial Fish Swarm Algorithm

Heyu Yang, Hongjun Li, Haiping Fan

School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai

Received: Mar. 8th, 2025; accepted: Apr. 15th, 2025; published: Apr. 25th, 2025

文章引用: 杨赫宇, 李洪军, 范海平. 基于改进人工鱼群算法的电商平台 AR-ASFA 分布式系统[J]. 软件工程与应用, 2025, 14(2): 346-358. DOI: 10.12677/sea.2025.142031

Abstract

The popularization of digitalization, informatization and the Internet has given birth to the rapid development of e-commerce. Faced with rapidly expanding user traffic, e-commerce platforms still face technical bottlenecks in carrying more user traffic and improving server response performance. This article optimizes the system from both software and hardware aspects to build a high-performance distributed AR-AFSA system. (1) The AR (Application Router) architecture is configured with three JobManager server nodes, which receive user access requests in three different ways. The traffic allocation mechanism is used to distribute the system's traffic carrying pressure, and user requests are divided into four traffic queues for scheduling according to different access methods. (2) Improved artificial fish swarm algorithm (AFSA) is used for container scheduling, re planning the execution order of various behaviors of artificial fish, increasing the possibility of finding the optimal solution and accelerating local convergence speed. (3) Parameters and evaluation indicators for artificial fish are designed to match containers with sufficient resources for user requests while ensuring resource conservation and system load balancing. Finally, validation was conducted on the Taobao user behavior dataset and multiple control experiments, and it was found that the AR system can withstand three times the traffic pressure of traditional servers. The improved artificial fish swarm algorithm can converge to a better solution compared to the control algorithm, and demonstrates greater advantages in situations where server resources are more complex.

Keywords

Distributed Architecture, Metaheuristic Algorithm, E-Commerce

1. 引言

AR-AFSA 系统是一个为用户提供高效率、低时延响应服务的电子商务平台系统，AR-AFSA 系统不仅在前端 JobManager 提高承受流量压力的能力，还通过启发式算法选择最适合处理用户请求的容器实例，在保证高性能处理请求的同时维护服务器系统的负载均衡。目前，大多数学者和工程师都通过优化程序算法和使用最新的开发框架优化系统的处理和响应性能，却忽略了系统在硬件架构和资源规划等方面的问题。

针对分布式架构中的服务器的流量承载问题，虽然通过集群、SOA 等架构方式优化了服务器的部署与设计架构，使服务器系统的架构更加稳定和可扩展，但是由于流量入口 JobManager 的负载上限问题，形成了木桶效应，即使服务器拥有更快的处理性能，但是一个公网 IP 只能对应一台物理机器，所以只能设置一台 JobManager 的方式使得系统的流量承载产生了瓶颈。因此在提升服务器系统承载流量的上限问题看似只能在硬件方面更换更高性能的 JobManager 服务器来解决。针对容器链中同种类实例的调度[1]-[3]问题，在系统接收到大量的请求后，除了在硬件方面使用更多更好的服务器来提高系统的性能，还应该在软件的算法中使用更好的设计，在采用一个合适的调度算法在很好地响应用户请求时，使服务器的负载更加均衡也是衡量一个算法优良性的重要评价指标。

由于单体软件架构过高的耦合性已无法满足当今超大规模发展的软件系统，随后产生的集群[4]和分布式[5]系统在硬件层面解决了大规模计算和高耦合的问题。之后面向服务的 SOA 架构[6]在分布式[7]-[9]

的基础上，将系统按服务继续细化，但是其高度的中心化导致和单体架构一样有无法快速拓展的缺点。在之前虚拟机的基础之上，容器技术出现，容器拥有着和虚拟机一样的虚拟资源，但是其不安装繁重的操作系统，启动速度快，轻量级的优点为后续的微服务系统架构[10]提供了轻型、组件化、兼容异构型、去中心化等的优良特性。

在物理机器配置方面，为了可以并行处理更多的用户请求，对于处理同种任务的容器，可以通过安装更多的容器实例来增加请求承载量。同时由于物理机器的大量增加，选择哪些实例投入运行是个重要的难题。软件规模的扩大导致的服务种类数增多，以及请求数量的增多，都会导致服务实例的规模不断增大。在选择服务实例时，需要综合考量请求的处理速度，系统资源的占用量，服务器间的负载均衡程度等指标。所以提升一个服务器系统的响应速度，不仅要在硬件层面对服务器进行升级更新，也要在算法中进行合理的优化，在软件和硬件两方面综合升级，才能构建一个优良的服务系统。

2. 系统架构

AR-AFSA 系统架构从硬件和软件两个方面对系统的服务效率进行综合优化，如图 1。硬件方面使用 AR 服务器架构通过设置多台 JobManager 节点服务器将用户流量进行分流，平衡 JobManager 单节点的承载压力。软件方面首先通过设置优先级队列，将用户提交的不同种类请求进行执行排序。然后从任务队列中提取第一个任务对其资源使用、所需交互数据等信息进行分析。分析得到其资源分配策略后，通过注册中心读取当前可执行任务的实例，并通过优化的人工鱼群算法计算出最优实例方案，最后放置在物理机中运行。

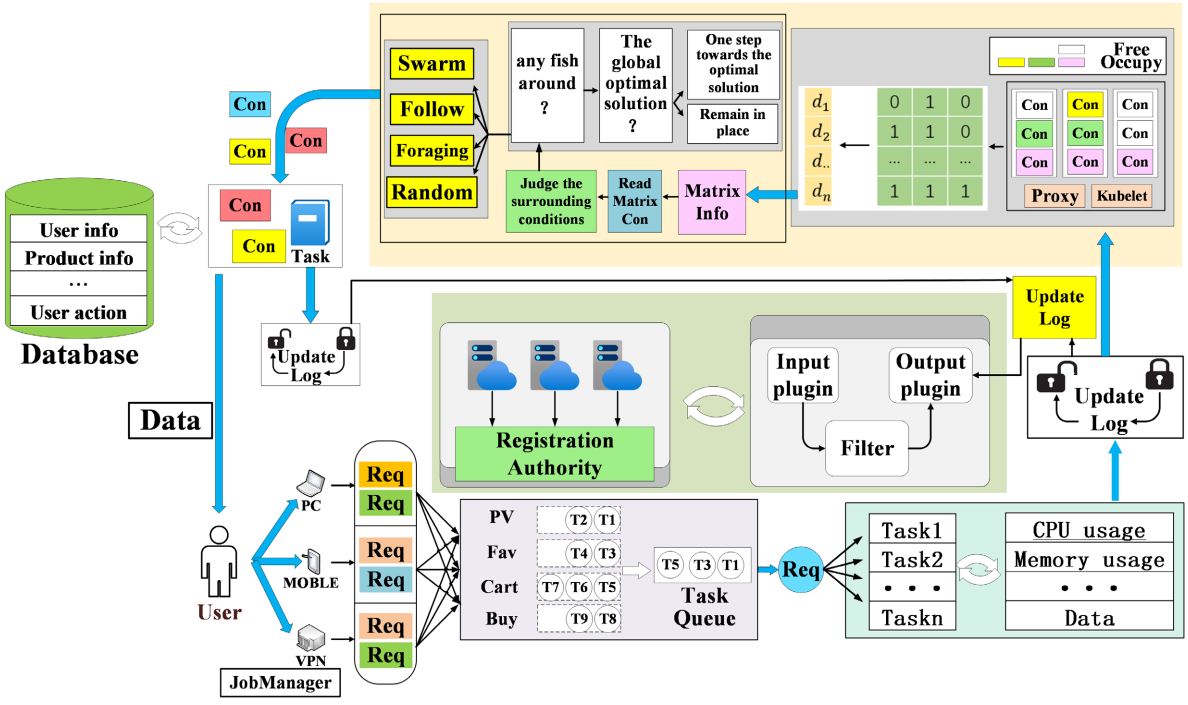


Figure 1. AR-AFSA system architecture
图 1. AR-AFSA 系统架构

2.1. AR 系统架构

传统的服务器架构系统中，JobManager 前端机统一接收所有用户访问请求，并根据业务逻辑进行处

理，在集群架构系统中，JobManager 识别请求后分配给不同的节点进行具体的业务处理，但是这种架构对 JobManager 的硬件性能要求非常高，且极易会出现响应时延的瓶颈。AR 架构在用户通过不同方式(PC 端、移动端、VPN 端)访问系统时，开发者在软件开发时就将该种类型软件所访问的 IP 地址写入程序，由于所使用的程序不同，在为不同种类型程序开发时，就写入了不同的 JobManager 前端机的 IP 地址，在终端设备的软件层上就已配置好需映射到的服务器集群。

2.2. 改进 AFSA 算法的准备

在电商系统中，四种用户操作：购买、收藏、加入购物车和访问，分别为这四种操作设置四个任务队列存放操作请求。并将如图 2 所示的方式将四种用户的操作设置为四种优先级，使用从优先级从高到低的轮询方式一次处理一个优先级的任务。同时兼顾四种用户请求的调度，可以兼顾四种用户的使用体验，既可以让使用量高的任务先执行，还兼顾了其他种类的任务请求，可以进一步发掘中长尾用户。

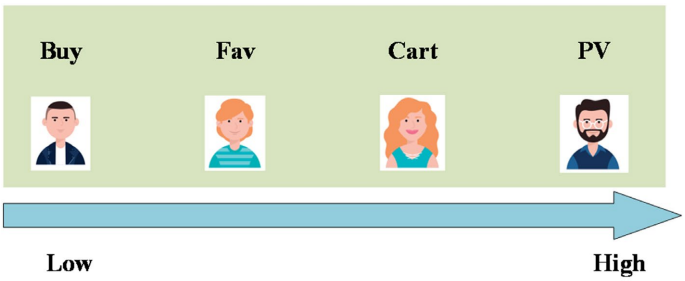


Figure 2. User operation priority
图 2. 用户操作优先级

由于分布式服务器架构是由多个物理机器组成，执行某种任务的容器实例数量可能有多个，将所有的容器实例集合认为是一个容器池，池中的容器实例具有相同或不同的 CPU、内存等物理资源配置。并通过优化的人工鱼群算法结合系统评判标准对实例的调度进行规划。容器池矩阵设计如下，每一行表示为一台物理机中，配置的多个容器节点，这些节点拥有一部分的物理资源；每一列表示不同物理机中处理相同种类任务的容器实例，由于每个实例在不同服务器中，而每台服务器的配置各不相同，所以池中处理相同任务的容器实例的计算速度和资源消耗量也不尽相同，这正是后文中启发式人工鱼群算法要解决的重要问题。

3. 模型

将抽象的任务转化为形式化的数学模型便于统一管理，数学模型的构建如下。首先用户提交的请求并经过分析后生成任务，由于任务由一组容器实例组成，所以用容器集 *ms set* 和容器间的关系 *ms relation* 来表示，其中 A 为一个需要执行的用户请求任务。

$$A = \langle ms\ set, ms\ \ relation \rangle \tag{1}$$

$$ms\ set = \langle ms_1, ms_2, \cdots, ms_n \rangle \tag{2}$$

$$(ms_1, ms_2) \in ms\ \ relation \tag{3}$$

用户任务需要执行的容器种类确定后，资源消耗量即可得出，这里的资源以 CPU 和内存两个计算资源作为计算尺度：

$$Resource_A = \langle CPU_{need}, Memory_{need} \rangle \tag{4}$$

为了确保所设计的系统是可行且高效的，需要使用一些计算模型来衡量系统的效率，来确保系统可以以更低的延迟运行，使用更少的资源运行，保证服务器集群的负载均衡防止某些服务器的性能过载而导致系统的低效运行，从时延 T、资源使用率 RC 和负载均衡度 LB 三个方面进行计算。三个指标的数值越低越好，所以最佳的解决方案应是当前方案中适应度值最低的方案。适应度值的计算由式(5)确定，设置权重 α 和 β 均为 0.5，其中 fit 表示适应度值，T 为任务所需时延，LB 为系统负载均衡度：

$$fit = \left(\alpha \times T + \beta \times RC\right) \times LB \tag{5}$$

在电商系统中，将用户分为访问、喜爱、加入购物车和购买四个类别，按照服务对象的重要程度，将其权重表示为 $\mu_1, \mu_2, \mu_3, \mu_4$，$fitness$ 表示最终适应度函数。

$$fitness = \mu_1 fit_{pv} + \mu_2 fit_{fav} + \mu_3 fit_{cart} + \mu_4 fit_{buy} \tag{6}$$

3.1. 时延

时延是一个系统运行速度的衡量标准，也是衡量一个系统效率的最重要的指标之一。系统的传输时延主要由节点的计算时延(CD)、传输时延(ND)和传播时延(PD)三部分组成：

$$T = CD + ND + PD \tag{7}$$

3.1.1. 计算时延(CD)

以上文设计的架构为标准，系统总时延包括各个容器结点计算的时间和在网络上的传输时间和传播时间之和。由于容器链具有串行、并行、复合等结构，每种结构的时延计算方式也不完全相同。传输时延在时延计算中也是不可忽略的因素之一，在异构的负载服务器集群中，数据会在节点中进行多次传输，这也会造成一定量的时延。

单节点的计算时延方式如下：

$$cd_i = Mask / capcity \tag{8}$$

运行总时延的计算公式如式(10)所示，需要将各段的处理时延相加得出总计算时延，各段的时延计算如式(9)，串行结构只需将各结点的计算时间及传播时间相加即可，是最简单的计算结构；并行结构需要统计每个分支的计算时延，并取最大计算时延(取并集)如式(9)。

$$Cd = \sum_{i \in tk} \max\left\{t_i\right\} \tag{9}$$

$$t_i = \sum_{j \in tc} cd_j \tag{10}$$

3.1.2. 传输时延(ND)和传播时延(PD)

数据在容器的各物理节点之间传输时，数据从发送方发送到信道上以及接收方将数据接受也是需要消耗一定的时间，如式(12)DV 表示要传输数据的数据量，BD 表示带宽。

除了计算和传输产生的基本时延外，数据在信道中传播的时延也是不可忽略的，其中 D_{ij} 表示节点 i 和节点 j 之间的距离，c 表示信号在信道中的传播速度：

$$ND = \sum_{x=1}^{m} \sum_{j=1}^{n} x_{ij} Nd_i \tag{11}$$

$$Nd_{ij} = \frac{DV_{ij}}{BD_{ij}} \tag{12}$$

$$PD=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}D_{ij}}{c} \tag{13}$$

3.2. 资源消耗

为计算任务的资源使用量，本节首先为 CPU 和内存的性能建立模型，如式(14)，R_{ij}^{CPU} 和 R_{ij}^{Mem} 表示 CPU 的计算能力和内存的容量，$Core_{ij}$ 表示容器的 CPU 的核心数，$sizeof\left(Core\right)$ 表示每个 CPU 核的计算能力，$Block_{ij}$ 表示容器所占内存的块数，$sizeof\left(Block\right)$ 表示每块内存的容量。

$$R_{ij}^{CPU}=Core_{ij}\times sizeof\left(Core\right) \tag{14}$$

$$R_{ij}^{Mem}=Block_{ij}\times sizeof\left(Block\right) \tag{15}$$

然后计算任务所消耗的 CPU 和内存资源数量，R_{t}^{CPU} 表示单位时间间隔 t 内容器的 CPU 使用成本。R_{t}^{CPU} 和 R_{t}^{Mem} 表示 CPU 和内存的单位时间 t 内的利用率，T_t 表示单位时间，CD_{ij} 表示应用的计算时间，然后与所占用的资源 R_{ij}^{CPU} 和 R_{ij}^{CPU} 相乘得出任务的资源总使用量。

$$RC_{ij}^{CPU}=R_{ij}^{CPU}\times\frac{CD_{ij}\times R_{t}^{CPU}}{T_t} \tag{16}$$

$$RC_{ij}^{Mem}=R_{ij}^{Mem}\times\frac{CD_{ij}\times R_{t}^{Mem}}{T_t} \tag{17}$$

最后将 CPU 和内存两个资源使用量相加得到该任务的资源总使用量，如式(18)：

$$RC=\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}RC_{ij}^{CPU}+\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}RC_{ij}^{Mem} \tag{18}$$

3.3. 负载均衡度

利用式(10)首先计算每台物理机的资源使用情况，通过物理节点的当前资源使用量与物理节点总资源量的比值 c_{ij}^{CPU} 获得：

$$c_{i}^{CPU}=\frac{\sum_{j=1}^{n}x_{ij}R_{ij}^{CPU}}{\sum_{j=1}^{n}R_{ij}^{CPU}} \tag{19}$$

对于一台物理机，使用 CPU 和内存的负载均衡度来评价一台物理机器的负载均衡度，并通过权重 ω 和 φ 来平衡两个评价指标，并且满足 $\omega+\varphi=1$。

$$L_{ci}=\omega c_{i}^{CPU}+\varphi c_{i}^{Mem} \tag{20}$$

下面计算出所有物理机器的平均负载均衡度：

$$\overline{L}=\frac{1}{i}\sum_{i=1}^{n}\left(\omega c_{i}^{CPU}+\varphi c_{i}^{Mem}\right) \tag{21}$$

最后通过计算所有物理机总体负载均衡程度的方差，理想的情况应该是保证各台物理机总体上都保持着相似的负载均衡程度，从而保证每一个节点的可用性，负载均衡程度(方差)越低越好：

$$LB=\sigma\left(L_1,\cdots,L_n\right)=\sum_{i=1}^{n}\frac{\left(L_{ci}-\overline{L}\right)^2}{k} \tag{22}$$

4. 改进的 AFSA 算法调度资源

由于服务器的资源以容器的方式被划分成多个拥有物理机部分资源来执行任务的容器实例，如图 3 所示为响应用户请求任务的多种容器组合方案，若采用不合理的实例组合处理任务，则会导致占用过多的物理机资源而导致其它任务的执行受阻或推后，故在容器池中找到适合本次任务执行的低时延、契合的资源和低负载均衡度的任务链是最为重要的，本节通过优化的启发式人工鱼群算法从一个多解的集合中，找到最符合设定的衡量标准的解。相对于其他求解方法，启发式算法在 NP-Hard 问题和动态规划问题中有着非常大的优势。

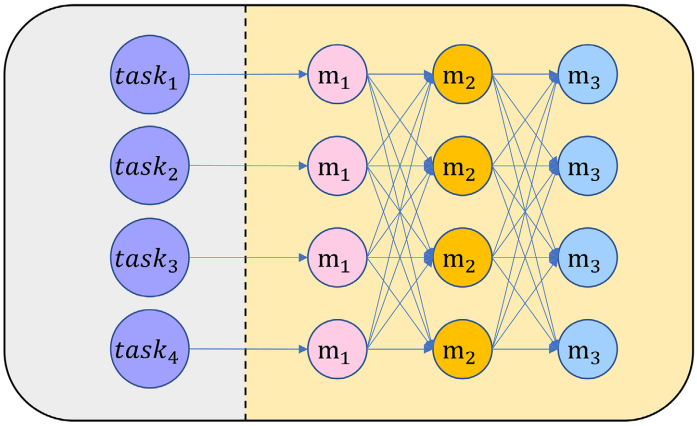


Figure 3. Container instance chain diagram
图 3. 容器实例链图

通过启发式的方法来解决容器的多目标优化类问题，人工鱼群算法是解决此类离散问题的一个非常好的方法。但是原始的人工鱼群算法过于简单，由于只在四种活动中做简单的适应度值比较后选择最优解，就简单完成一次迭代，不能很好地利用人工鱼对最优解进行高效探测。在极端的情况下，例如解空间范围较大，并且初始鱼群分布位置较分散时，人工鱼无法群聚和觅食行为，大部分的鱼只能盲目地随机觅食且无方向的随机游动，最后使计算结果偏离最优解。在人工鱼视野内没有其它鱼时，会选择随机方向进行觅食行为，在找到有更好位置的情况下就进行游动，由于过度的随机性会导致人工鱼的单次行动是更优的，却在全局可能会偏离最优解的方向。同时过度的随机性也会导致算法每次执行时的结果参差不齐，降低算法的整体优良性，因此本章根据全局最优解作为预测目标为本次求解提供大致搜索方向。

本节对启发式人工鱼群算法的步骤进行完善和优化。一个执行的任务即为一条容器链，一条容器链由多个容器组成，人工鱼个体代表一个解即一个任务的解决方案，人工鱼个体位置的变化代表在此条容器链中某个或某些容器选择策略的改变。首先对人工鱼的步长、视野范围、人工鱼数量、迭代次数、拥挤度因子等参数进行初始化。然后判断人工鱼的视野范围内是否有其它的鱼，若有其它鱼则进行群聚或追尾行为，否则就向全局位置最好的鱼即当前全局最优解移动一个步长的距离，若此条人工鱼为全局最优解，则此次停止行动，继续其它人工鱼的活动。在群聚行为中，人工鱼会向视野范围内的鱼群中心游动，为了防止大量人工鱼聚集在一起，而陷入在局部最优解中，首先预设一个鱼群的拥挤度因子，在人工鱼进行群聚行动之前，先检查鱼群的拥挤度因子，若未超过因子则说明鱼群还不够拥挤，则向鱼群中心行动；若超过阈值表示鱼群拥挤，就进行随机行为。在群聚和追尾两种行为判断结束后，通过比较两种预测行为更新位置后的最好适应度值，取出并与人工鱼当前位置的适应度值进行比较，若适应度值更好则更新位置，否则不更新。算法描述如下：

改进的人工鱼群算法(AFSA)

Input: *Pop number, N_max, try_number, step*
Output: *fitness_list[]*
for each T **do**
N ← 1
Pop ← Initialize Population()
Calculate the fitness values;
front ← CaleulateFronts(*Pop, fitness*);
while n <= N do
for each Pop_i **do**
$distance_{ij}$ = $distance_i$- $distance_j$;
end for
if $distance_{ij}$<visual **then**
if is not the ground best **then**
go one feet to the ground best;
else
take a step according to the forecast result;
end if
else
best ← Min(*best, Swarm()*);
best ← Min(*best, Follow()*);
best ← Min(*best, Prey()*);
end if
end while
calculate the best fitness;
best ← thebest;
end for

5. 实验验证

为了验证 AR 架构的优良性和优化的人工鱼群算法在容器实例调度的优良性以及相比传统群体智能算法有更高的效率。在 5.1 节中对传统架构中单 Jobmanager 与 AR 架构中多 JobManager 的响应流量数量进行了比较。在 5.2 节中分别在物理机数量相同，容器数量不同，以及每台机器容器数量相同，物理机数量不同的情况下，比较 AR-AFSA 算法与对照算法的系统资源使用量。5.3 节中 5.2 节相同的设置条件下比较 AFSA 算法与对照算法的综合成本分析。最后验证了优化后的人工鱼群算法在运行效率上的高效性以及优化的算法在容器实例调度中的有效性。

实验环境为 64 位操作系统，系统版本为 Windows11，处理器为 i5-12500h (12 核 16 线程)，主频 2.5GHz，16GB DDR4 内存。AR 架构的实验使用 python 的 flask 框架，优化的人工鱼群算法通过 matlab 进行仿真验证。

5.1. AR 架构性能分析

为了评估 AR 架构的性能，本节基于第二章中的架构设计了实验，第一阶段实验依靠所设计的 AR 架构，对服务器架构进行压力测试，传统架构使用一台 JobManager 作为用户请求接收服务器，所以根据实际场景实验也同样采用一台物理机作为 JobManager 进行压力测试。实验环境使用基于 Python 的 Flask 服务器框架，通过日志输出查看节点可承受的访问量，实验记录 1 秒内相应的服务请求次数并作为服务器的承载压力极限，将所有请求发送至该节点。取代单 JobManager 节点，AR 架构以三台 JobManager 为例，设置三台 JobManager 服务器性能相同，实验方法与上述大致相同，实验结果如图 4。与上述实验相似，取一整秒的服务器响应情况来表示该节点可承载的最大访问压力。

通过实验可得，在原始的单 JobManager 节点架构中，若三个服务器集群使用一个 JobManager 接受

网络请求，将所有的请求集中在一个节点上，在 JobManager 服务器的配置及参数相同的情况下，那么系统的吞吐量只有 AR 架构的三分之一。所以通过实验也验证了在理论上 AR 架构可以达到原始架构三倍吞吐量的性能。

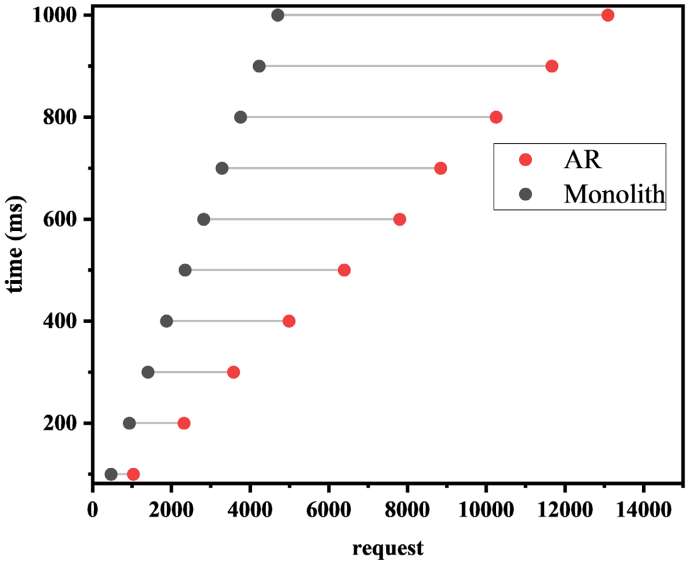


Figure 4. Comparison of architecture carrying traffic
图 4. 架构承载流量比较

5.2. 数据集分析

实验中的数据使用阿里云淘宝用户购物行为 UserBehavior 数据集，数据集中共有 100,150,807 条用户行为数据，用户行为有访问、加入购物车、收藏和购买四种行为，取连续 200 万条数据进行实验。得到用户的行为分布如图 5 所示，对于用户行为的比例分析，为平衡四种行为对于适应度值的影响，我们将 $\mu_1, \mu_2, \mu_3, \mu_4$ 分别设置为 0.02，0.03，0.055，0.9。

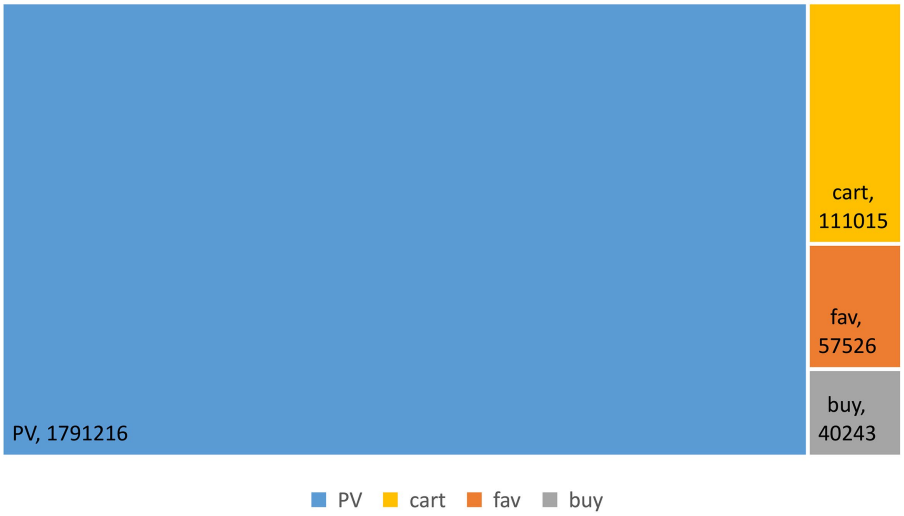


Figure 5. Analysis of user behavior
图 5. 用户行为分析

5.3. 资源使用分析

Table 1. Parameter settings for algorithms
表 1. 算法的参数设置

算法	参数	值
PSO	Inertia weight	0.6
	Acceleration constant	2
GA	Crossover probability	0.7
	Mutation probability	0.01
AR-AFSA	Vistial	2.5
	Step	0.3
	Delta	0.618
	Trynumber	50

Table 2. Parameter settings under different number of instances
表 2. 实例数量不同下的参数设置

Experiments	Machine	Resource scope	Instances	Interval
Experiment1	2	(0,20]	400	0.1
Experiment2	2	(0,30]	600	0.1
Experiment3	2	(0,40]	800	0.1
Experiment4	2	(0,50]	1000	0.1

Table 3. Parameter settings under different numbers of physical machines
表 3. 物理机数量不同下的参数设置

Experiments	Machine	Resource scope	Instances	Interval
Experiment1	2	(0,20]	400	0.1
Experiment2	3	(0,20]	600	0.1
Experiment3	4	(0,20]	800	0.1
Experiment4	5	(0,20]	1000	0.1

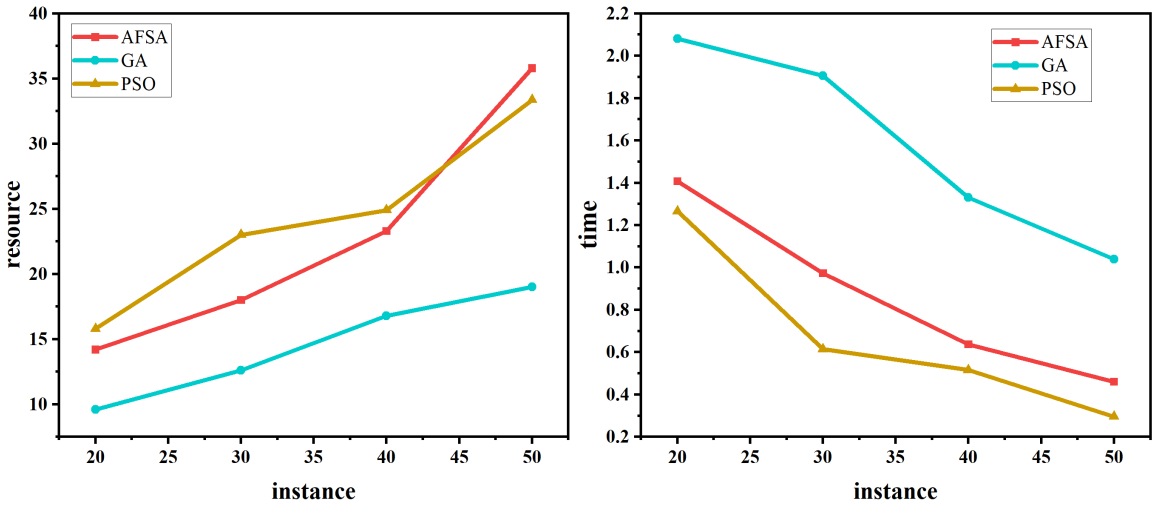


Figure 6. Resource and time consumption under different numbers of container instances
图 6. 不同容器实例数量下的资源和时间消耗

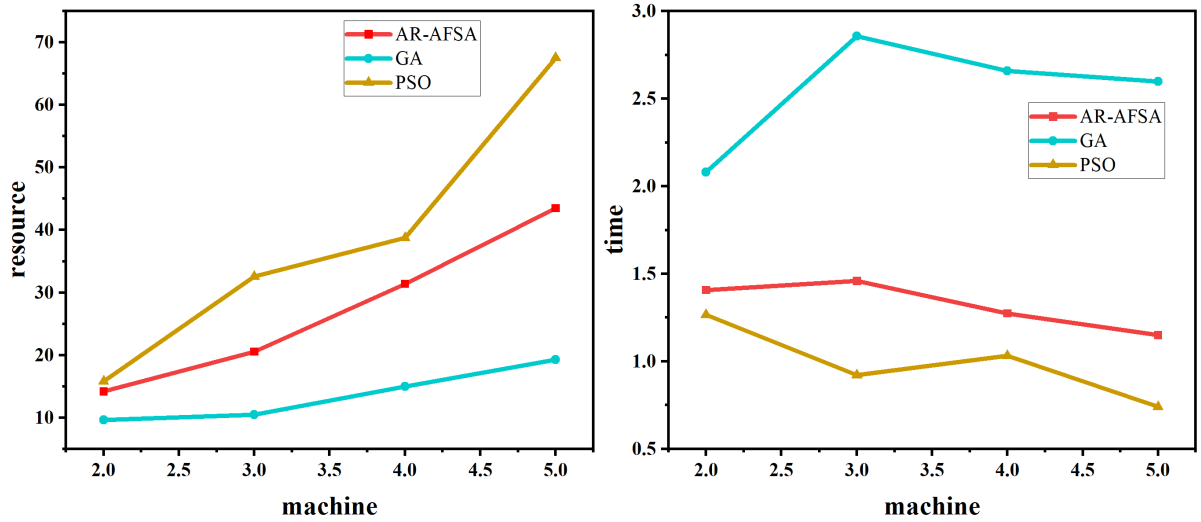
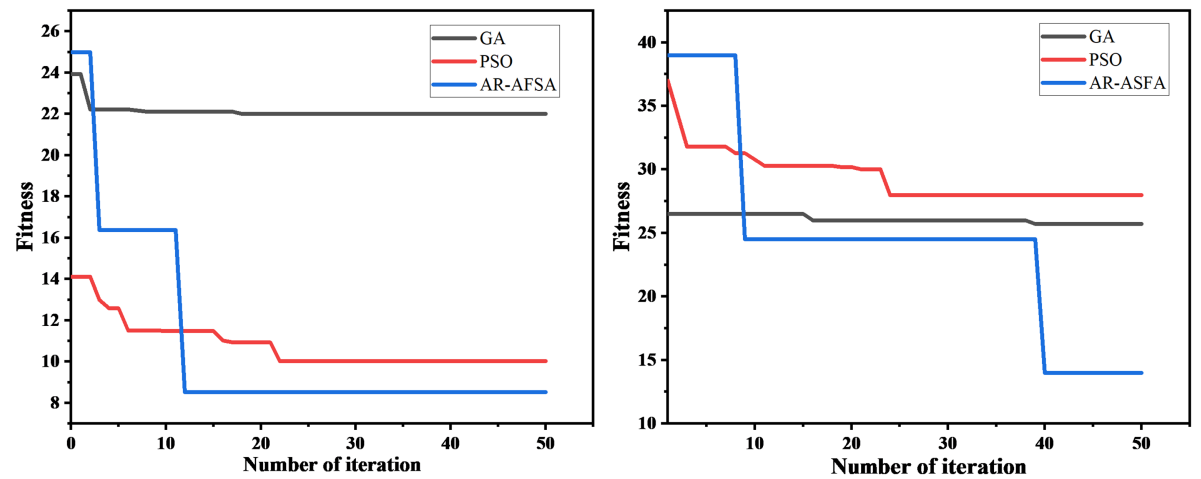


Figure 7. Resource and time consumption under different numbers of physical machines
图 7. 不同物理机数量下的资源和时间消耗

实验参数设置如表 1 所示，两组实验设置如表 2 和表 3 所示。如图 6 和图 7 所示，在与对照算法 PSO 和 GA 中，AR-AFSA 处在居中的资源使用量，同时处理请求的时延最低。系统容器的实例数量与资源的使用量成正比，随着容器数量或虚拟机数量的增加，算法使用 CPU、内存资源使用量逐渐增加，系统处理用户请求的时延逐渐变短。可以解释为，在系统提供了更多的容器选择方案后，系统选择了资源更多的容器进行请求处理，同时所选容器性能的提高减低了处理的时间。改进的 AFSA 在一开始并未展示出最佳的性能，这是因为在环境规模较小的情况下，改进的 AFSA 较多的优化步骤降低了算法的求解速度。

5.4. 综合成本分析

对优化后的人工鱼群算法的优良性进行测试，实验对照设置与上节相同。为测试提出的 AR-AFSA 算法的拓展性，如图 8 为实验结果，表示容器链选择方案的优良性以式(5)评价。在物理机数量保持不变下，逐渐增加每台物理机中实例的数量。虽然 AR-AFSA 算法在随机产生的初始解的适应度值略高于对照算法，但在迭代过程中算法以更短的迭代次数找到最优解，这是因为改进的 AFSA 人工鱼在迭代时通过预测和全局最优解找到了更优解，同时 AR-AFSA 算法在找到的最优解的适应度值低于对照算法。



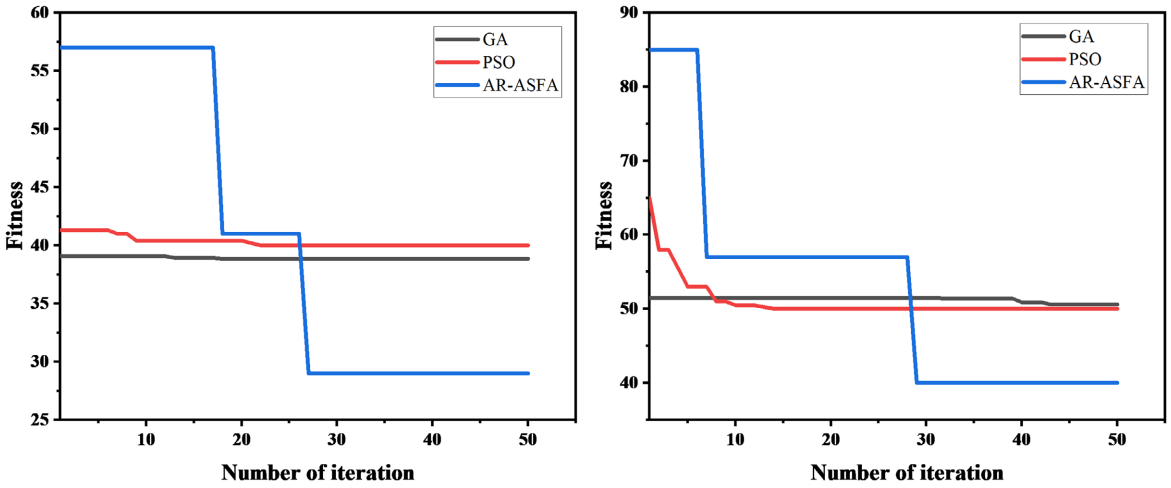


Figure 8. Performance comparison of algorithms under different instance numbers
图 8. 不同实例数下各算法的性能比较

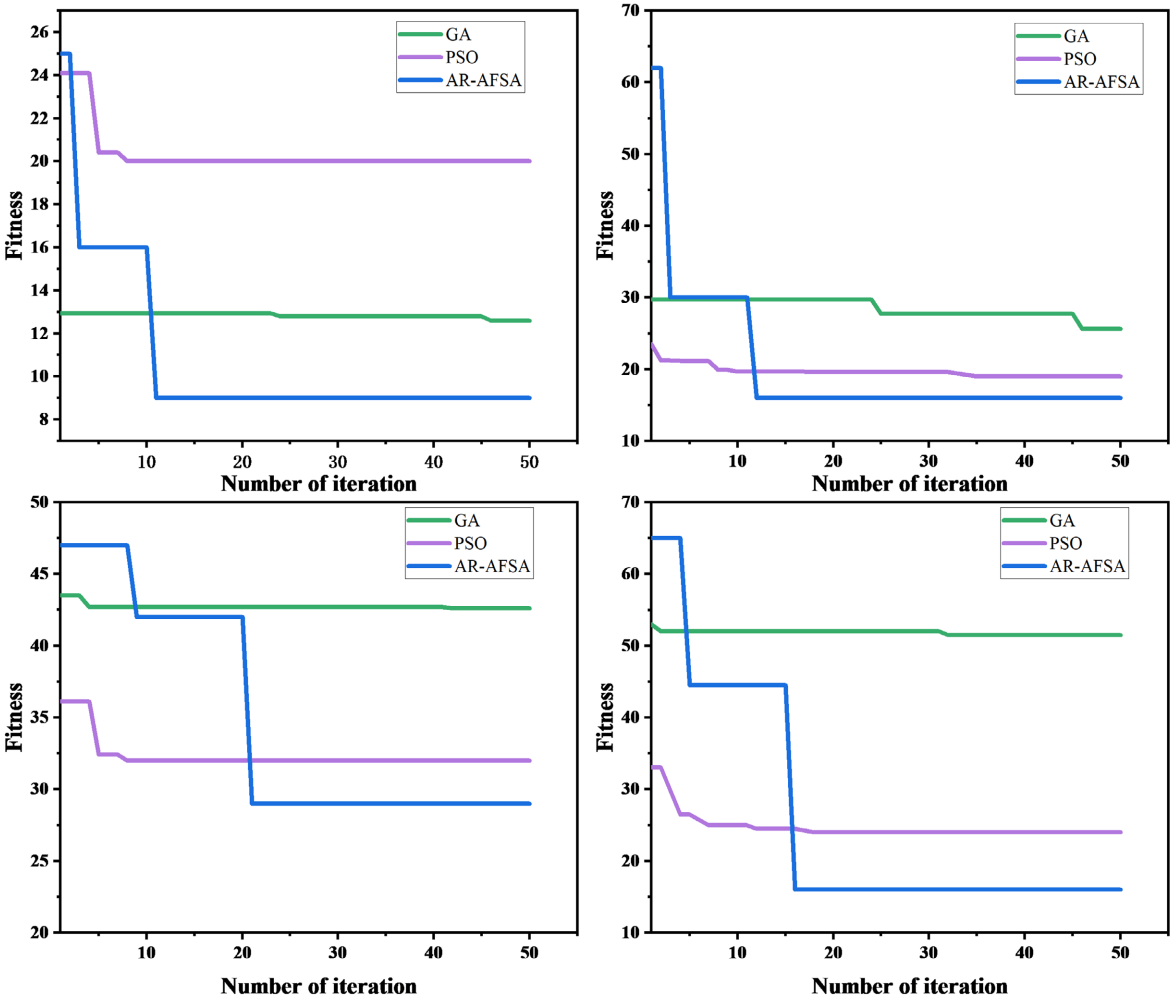


Figure 9. Performance comparison of algorithms under different numbers of physical machines
图 9. 不同物理机数量下各算法的性能比较

继续对优化后的人工鱼群算法进行验证，保持每台物理机的容器实例数量不变，设置四个实验分别使用不同数量的物理机进行实验。如图 9 所示，随着物理机器数量的逐渐扩大，任务所需实例数量增多，导致算法随机初始化的适应度值越高，这是因为要完成的任务增多导致需要的容器实例数在增多，所以所需的资源量和处理时延也在增加，所需的迭代次数也就会越来越大。经过对比，提出的 AR-AFSA 算法可以应对本节实验设计的物理机数量变化的情况，AR-AFSA 算法在物理机数量增加时，依然可以保持较低和较稳定的适应度值，并且优于其余对照算法。

6. 总结

实验结果表明，AR 架构的流量承载量为传统服务器架构的 3 倍。在容器资源调度方面，改进的 AFSA 在设定的测试指标下，对容器资源调度的效率和搜索方案都优于对照算法。AR-AFSA 架构相对于传统的单体 JobManager 服务器架构以及传统的启发式算法性能更佳，具有更多优势。

AR-AFSA 是一个在软硬件综合性能优化的电子商务分布式服务器系统。针对电子商务中的高并发流量导致服务器过载的问题，本文通过 AR 架构配置多台 JobManager 服务器节点通过应用层进行流量分流映射；通过改进人工鱼群算法，保证完成响应的情况下进行容器实例的调度。

本研究不仅可以在电子商务系统中提升用户的使用体验和系统服务质量，还可以应用在医疗、餐饮等行业进行应用和推广。未来，将 AR-AFSA 系统与用户群体分类结合，结合大量用户数据，构建更完善的电商系统。

参考文献

[1] Niu, W. and Li, J. (2022) A Two-Stage Cooperative Evolutionary Algorithm for Energy-Efficient Distributed Group Blocking Flow Shop with Setup Carryover in Precast Systems. *Knowledge-Based Systems*, **257**, Article 109890. https://doi.org/10.1016/j.knosys.2022.109890

[2] Li, C., Liu, J., Li, W. and Luo, Y. (2021) Adaptive Priority-Based Data Placement and Multi-Task Scheduling in Geo-Distributed Cloud Systems. *Knowledge-Based Systems*, **224**, Article 107050. https://doi.org/10.1016/j.knosys.2021.107050

[3] Liu, G. (2023) A Q-Learning-Based Distributed Routing Protocol for Frequency-Switchable Magnetic Induction-Based Wireless Underground Sensor Networks. *Future Generation Computer Systems*, **139**, 253-266. https://doi.org/10.1016/j.future.2022.10.004

[4] Ohi, A.Q., Mridha, M.F., Safir, F.B., Hamid, M.A. and Monowar, M.M. (2020) Autoembedder: A Semi-Supervised DNN Embedding System for Clustering. *Knowledge-Based Systems*, **204**, Article 106190. https://doi.org/10.1016/j.knosys.2020.106190

[5] Singh, H.J. and Bawa, S. (2022) Lameta: An Efficient Locality Aware Metadata Management Technique for an Ultra-Large Distributed Storage System. *Journal of King Saud University-Computer and Information Sciences*, **34**, 8323-8335. https://doi.org/10.1016/j.jksuci.2022.08.012

[6] Costa, B., Pires, P.F. and Delicato, F.C. (2020) Towards the Adoption of OMG Standards in the Development of Soa-Based IoT Systems. *Journal of Systems and Software*, **169**, Article 110720. https://doi.org/10.1016/j.jss.2020.110720

[7] Li, J. (2020) Resource Optimization Scheduling and Allocation for Hierarchical Distributed Cloud Service System in Smart City. *Future Generation Computer Systems*, **107**, 247-256. https://doi.org/10.1016/j.future.2019.12.040

[8] Yin, L. and Sun, Z. (2022) Distributed Multi-Objective Grey Wolf Optimizer for Distributed Multi-Objective Economic Dispatch of Multi-Area Interconnected Power Systems. *Applied Soft Computing*, **117**, Article 108345. https://doi.org/10.1016/j.asoc.2021.108345

[9] Moghadam, A.S., Suratgar, A.A., Hesamzadeh, M.R. and Nikravesh, S.K.Y. (2022) Multi-Objective ACOPF Using Distributed Gradient Dynamics. *International Journal of Electrical Power & Energy Systems*, **141**, Article 107934. https://doi.org/10.1016/j.ijepes.2021.107934

[10] Cinque, M., Della Corte, R. and Pecchia, A. (2022) Micro2vec: Anomaly Detection in Microservices Systems by Mining Numeric Representations of Computer Logs. *Journal of Network and Computer Applications*, **208**, Article 103515. https://doi.org/10.1016/j.jnca.2022.103515