计算机工程
Computer Engineering
ISSN 1000-3428,CN 31-1289/TP

《计算机工程》网络首发论文

题目：微服务架构 FPGA 云平台的并发请求调度机制研究
作者：奚智雯，蔡晶晶，阳文敏，柴志雷
DOI：10.19678/j.issn.1000-3428.0062678
网络首发日期：2021-11-05
引用格式：奚智雯，蔡晶晶，阳文敏，柴志雷．微服务架构 FPGA 云平台的并发请求调度机制研究[J/OL]．计算机工程.
https://doi.org/10.19678/j.issn.1000-3428.0062678

计 算 机 工 程
Computer Engineering

微服务架构 FPGA 云平台的并发请求调度机制研究

奚智雯[1]，蔡晶晶[1]，阳文敏[2]，柴志雷[1,3]

(1.江南大学，人工智能与计算机学院，江苏 无锡 214122；2.江苏虎甲虫计算技术有限公司，江苏 无锡 214000；3.江苏省模式识别与计算智能工程实验室，江苏 无锡 214122)

摘 要：针对基于微服务架构的 FPGA 云平台存在的大量用户并发请求的处理问题，提出了一种基于优先级调度的自定义参数响应指数计算模型。基于该模型设计了一种高并发请求（HCRS）调度算法，通过响应指数阈值对请求类别进行划分，保证最高优先级请求优先得到处理，次优先级请求加入先进先出队列等待，低优先级请求暂时挂起。缩短请求响应时间，缓解高并发请求带来的资源分配压力。通过在真实运营的 FPGA 公有云平台中实现该算法并在真实环境中进行测试，实验结果表明：在并发请求量相同时，HCRS 算法与先来先服务(FCFS)算法相比，平均响应延时降低 29.074s；平均请求响应时间降低 12.605s；该算法同时还有助于提升相应处理的吞吐量、系统并发度，可以有效提升节点资源的利用率。

关键词：微服务架构；FPGA；并发请求；服务质量；云计算；优先级调度

开放科学（资源服务）标志码（OSID）：

A Study of Concurrent Request Scheduling Mechanism for an FPGA Cloud based on Microservice Architecture

Xi Zhiwen[1],Cai Jingjing[1],Yang Wenmin[2],Chai Zhilei[1,3]

(1.School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi, Jiangsu 214122, China;

2.Jiangsu Tiger Beetle Computing Technology Co., Ltd , Wuxi, Jiangsu 214000, China;

3. Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence , Wuxi, Jiangsu 214122, China)

【Abstract】The FPGA cloud platform based on microservices architecture has a problem with a large number of user concurrent requests. To address this problem, this paper proposes a calculation model of user-defined parameter response index based on priority scheduling. Then, a high concurrent request scheduling algorithm is designed based on this model. This algorithm classifies requests by response index threshold. Therefore, the highest priority requests are processed first, the secondary priority requests are added to the first-in-first-out queue to wait, and the low priority requests are temporarily suspended. This method shortens the request response time and eases the resource allocation pressure caused by high concurrent requests. Experiments compared the performance of the HCRS algorithm with the FCFS algorithm on the FPGA public cloud platform in a real scenario. The average response latency of the HCRS algorithm is reduced by 29.074s and the average request response time is reduced by 12.605s. The proposed contribution helps to improve throughput, system concurrency, and effectively improves the utilization of node resources.

【Key words】microservice architecture; FPGA; concurrent request; QoS; cloud computing; priority scheduling

DOI:10.19678/j.issn.1000-3428.0062678

0 概述

现场可编程门阵列（Field Programmable Gate Arrays，FPGA）是一种可由用户定义其硬件架构的“空白芯片”，广泛应用于专用硬件设计、芯片验证领域，并大量用于计算机组成原理、集成电路设计等课程实验[1,2]。但传统的基于 FPGA 的实验通常依赖于专用设备，需要学生指定时间到指定地点进

基金项目：国家自然科学基金（61972180）

作者简介：奚智雯（1997-），女，硕士研究生，主研方向为计算机体系结构；蔡晶晶，硕士研究生；阳文敏，硕士；柴志雷，教授、博士。**E-mail**：**zlchai@jiangnan.edu.cn**

行，影响了实验教学和系统开发的效率。随着互联网技术的发展，出现了不同类型的 FPGA 云平台，致力于推动FPGA生态在更大范围内开放[3-5]。FPGA 云平台的系统架构也由耦合度较高、灵活性较差、开发运维成本较大的单体架构逐渐向松耦合、可伸缩、易部署的微服务架构转变[6]。微服务最初由 James Lewis 和 Martin Fowler 定义，是对传统单体架构的解耦，其将功能拆分成多个不同的服务，便于设计、开发以及运维部署[7]。随着用户请求量的激增，FPGA 云平台面临高并发请求的应用场景。由于单体架构的调用请求都在本地，所以其网络延迟较低，但在高并发请求下，微服务架构的调用可以同时进行，且不相互依赖，总执行时间也越来越短，相较于单体架构性能显著提高[8]。

目前，许多研究人员对微服务架构的云平台并发请求进行了相关研究。这些研究分为两个方面，一方面侧重于保证服务质量 QoS（Quality of Service）[9]，如 Shetty 等人提出了一种具有服务分组的云服务选择机制，基于改进的层次分析法分组选择算法来提高性能[10]，但其仅适用于共享服务调度系统；Mengyu 等人提出了一种支持并发请求的物联网服务组合优化节能机制，该技术可以提高物联网服务在并发请求间的可共享性[11]，但其受限于不同服务之间的粒度影响；郭等人为提高整体的 QoS 提出一个主动式突发并发量应对策略 GMAC，该策略能预测用户并发量[12]，但并未进一步作出突发并发量的应对策略。另一方面从负载均衡入手深入研究，如 Amal 等人为保证资源的有效使用提出一种负载均衡算法，对并发请求用户进行优先级排序[13]，Liu 等人通过确定综合负载值，从而衡量资源的利用率[14]，但都未在真实系统中处理并发请求；Zhou 等人提出了一种基于微服务的边缘计算的负载建模方法[15]，是优化资源配置和调度的前提，但这种方法并不通用，局限于特定类型的应用。

上述研究提及的相关算法通过减少选择服务的计算时间和减少重复的存储库访问来提高并发性能，仅适用于多服务共享调度系统，而 FPGA 云平台是基于单服务的调度系统，无需进行服务选择。因此，上述算法不适用于处理 FPGA 云平台的并发请求问题。

本文基于单服务调度系统下的 FPGA 云平台建立自定义参数响应指数计算模型，用于计算高并发请求下每个用户对应的响应指数。基于该模型，设计一种高并发请求调度（HCRS）算法。最终在真实运营的 FPGA 云平台中使用同一组用户数据进行测试，实验结果表明：在相同并发量下，本文算法与先来先服务（FCFS）调度算法相比，平均响应延时降低 29.074s、请求响应时间降低 12.605s，同时该算法还有助于提升响应处理的吞吐量、系统并发度，可以有效提升节点资源的利用率。

1 基于微服务架构的 FPGA 云平台

微服务架构是按照业务边界做细粒度的拆分以及部署，将一个大型且复杂化的单体架构应用拆分出多个松耦合的微服务[16]。因此，本文在设计 FPGA 云平台时根据实际业务需求将单体架构模式下的平台拆分出节点资源管理这一核心功能作为独立的一个微服务，其特点是能够随时拥有整个集群的使用状况，并且能以全局视角选择合适的节点进行分配任务。微服务通过轻量级通信框架 REST（Representational State Transfer）与外界进行通信[17]，同时支持使用不同技术或者平台实现。

1.1 FPGA 云平台架构

FPGA 云平台主要由 Web 应用层以及节点资源管理微服务组成，其系统架构及关键组件如图 1 所示，终端用户可以在不同设备上使用网络浏览器访问 FPGA 云平台，在云平台上可以进行申请使用节点，检查 FPGA 硬件节点实时执行状态等操作。

其中，Web 应用层含有心跳机制[18]，该心跳机制用于检查用户申请到的节点是否长时间处于未使用状态，若用户在一段时间内未发送心跳包，则自动释放节点，未使用的这段时间即为用户空置时间。而节点资源管理

微服务主要负责管理节点的状态。二者均使用 ELK（Elasticsearch，Logstash，Kibana）[19]日志平台进行日志采集与分析，并通过 RESTful API 进行通信，客户端发起请求经过 Nginx（反向代理服务器）处理；对微服务接口添加 JWT（Json web token）访问认证机制，保证接口的安全性；并对节点使用界面 Jupyter notebook 配置安全策略，保障硬件节点使用安全。数据存储层则使用关系型数据库 Mysql 以及非关系型数据库 Redis。

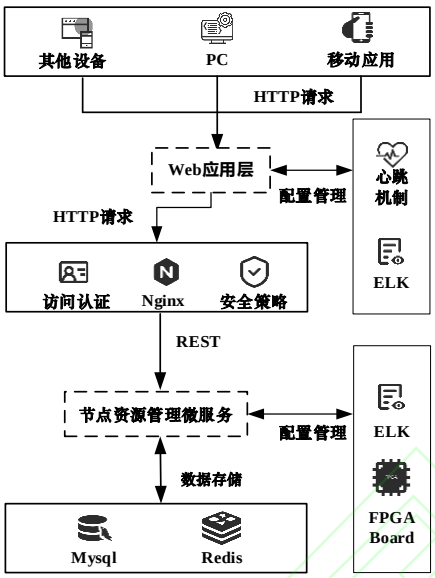

图 1 FPGA 云平台系统架构

Fig.1 FPGA cloud platform system architecture

1.2 节点资源管理微服务

节点资源管理微服务包含六个主要功能，具体如下：服务注册发现、节点申请、节点释放、节点实时状态查询、重置节点系统、定时测试节点等。微服务架构如图 2 所示，底层采用 Python 编写的 Web 应用框架 Tornado（Tornado Web Server），控制层为 RM Server 节点状态控制器。微服务为节点定义了六种状态，分别为：Boot（初始态）、Testing（测试态）、Valid（可用态）、Busy（占用态）、Resetting（重置态）、Broken（故障态），节点状态控制器的作用即为管理这六种状态的逻辑转换。其中，在申请和释放这两个操作时需加入互斥锁来解决资源同步与互斥问题。业务逻辑层为各个上述主要功能的具体实现，对主要功能进行实现与封装，并将封装好的服务以接口的形式利用通信层 RESTful API 与 Web 应用层进行通信，便于 Web 应用层调用接口。

RESTful API		
Server Login-JWT	Apply-Iptables, mount	Release-Iptables, umount
Check Node status	Testing Op-Rsync, XSDB rst	Rsync Node-Rsync
RM Server-Nodes status controller		
Tornado		

图 2 节点资源管理微服务架构

Fig.2 Node resource management microservice architecture

1.3 并发请求应用场景描述

在微服务架构下的 FPGA 云平台实际应用场景中，由于 FPGA 服务器硬件节点个数有限，当大量用户在同一时间并发申请节点时，过程如图 3 所示，请求响应时间较长且无法满足所有用户的请求，微服务节点资源被占满甚至直接导致微服务宕机。因此，针对用户申请节点高并发请求问题展开研究。

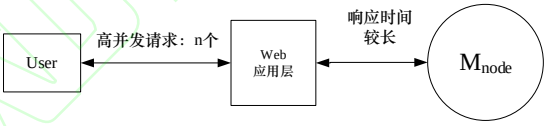

图 3 高并发请求过程

Fig.3 High concurrent request process

2 节点资源管理微服务高并发请求调度算法

本文从节点资源管理微服务的高并发请求实际应用场景出发，建立自定义参数响应指数计算模型，确保能够准确描述每个用户申请节点请求的优先级顺序。在此基础上，提出高并发请求（HCRS）调度算法，实现节点资源合理分配，并且优化微服务质量。

2.1 参数设置及权重确定

2.1.1 参数设置

为解决在 FPGA 云平台上用户申请节点高并发控制问题，从优先级调度入手，分析用户申请操作的关键影响因素，将这些关键因素作为自定义参数，具体定义如表 1 所示。其中，用户计划使用时间表示被申请节点预计占用时长，历史使用平均时长则用来评估用户历史使用频率，而历史空置时间则用来衡量用户是否高效使用节点。

表 1 申请节点自定义参数

Table 1 User-defined parameters of applying for nodes

符号	定义
T_p	用户计划使用时间
V	是否是会员
N_i	用户计划申请节点数量
H_{avg}	用户历史使用平均时长
T_e	用户历史空置时间

2.1.2 自定义参数权重确定

本文采用 AHP（层次分析法）确定自定义参数的权重。该方法是一种定性和定量分析相结合的决策方法，具有系统、灵活及简洁的优点。在解决实际问题时，层次分析法处理问题的思路与人的决策思维较为相似。AHP 有四种计算方法求权重，而四种计算方法得出的权重向量一般比较接近，仅有细微差别[20]，故本文选取其中一种方法即算术平均法计算权重。

表 2 重要程度及其对应数值

Table 2 Importance and its corresponding value

标度	含义
1	同等重要
2	同等到中等重要
3	中等重要
4	中等到很重要
5	很重要
6	很重要到非常重要
7	非常重要
8	非常重要到极其重要
9	极其重要
倒数	若因素 i 与 j 的重要性之比为 a_{ij}，那么因素 j 与因素 i 重要性之比为 $a_{ji}=1/a_{ij}$

结合本文应用场景，AHP 可以根据各个自定义参数的重要级别对其进行排名，从而得出参数的权重大小。层次分析法的重要程度及其对应数值如表 2 所示。

表 3 判断矩阵

Table 3 Judgment matrix

自定义参数变量	T_p	V	N_i	H_{avg}	T_e
T_p	1	1/3	1	1/3	1/2
V	3	1	3	2	2
N_i	1	1/3	1	1/2	1
H_{avg}	3	1/2	2	1	1
T_e	2	1/2	1	1	1

根据表 2 所示比较标准，对上述自定义的五个参数：用户计划使用时间、是否是会员、用户计划申请节点数量、用户历史使用平均时长以及用户历史空置时间进行互相比较，得出如表 3 所示的判断矩阵。

根据判断矩阵[21]，计算各个参数对应的权重，本文使用 Python 实现了层次分析法分析的完整过程，首先对判断矩阵进行一致性检验，检验通过方可进行权重计算，AHP 计算权重算法具体如下：

算法 1 AHP 计算权重算法

输入 判断矩阵

输出 参数权重大小

```
1. AHP(criteria);
2. max_eigen,CR,criteria_eigen ← weight_func(criteria);
3. if CR < 0.1 //一致性检验
4. else
5. end if;
```

经过层次分析法求得各参数权重具体如下：W_1=0.1，W_2=0.4，W_3=0.1，W_4=0.2，W_5=0.2。

2.2 模型建立

2.2.1 响应指数函数定义

FPGA 云平台在上线运行过程中，面临海量用户的注册登录以及申请节点等操作时，就会产生响应时间慢，响应延时较长以及长时间申请不到节点等问题。为解决上述问题，本文设计并建立了一个自定义参数响应指数计算模型，模型主要用于对每个用户的申请输入参数进行分析计算，对应的申请节点请求响应指数公式如式(1)所示：

$$F(y)=W_1*X_{\text{plan}}+W_2*X_{\text{vip}}+W_3*X_{\text{nodenum}}+W_4*X_{\text{hisavg}}+W_5*X_{\text{empty}} \quad (1)$$

其中，y 为响应指数，W_1 到 W_5 即为参数对应的权重，X_{plan}，X_{vip}，X_{nodenum}，X_{hisavg}，X_{empty} 分别为用户计划使用时间、是否是会员、用户计划申请节点数量、用户历史使用平均时长以及用户历史空置时间归一化后的值。

2.2.2 响应指数阈值定义

该模型模拟用户通过 Web 应用层发起申请节点请求，传递相应的请求参数后，需要根据响应指数阈值对所有用户进行区分，而设置最佳上限和下限阈值显得极

为重要，其有助于实现微服务的自动伸缩[22]。因此，本文模拟了 5 组不同数量的用户并发登录及申请节点请求并分别计算其响应指数，并使用 IBM SPSS Statistics 开源工具对数据进行分析，具体阈值数据分析如表 4 所示，其中α表示平均响应指数，β表示最低响应指数。据表 4 分析可知，五组数据均保留小数点后一位，可得平均响应指数α=0.2，最低响应指数β=0.1。

表 4　阈值分析表

Table 4　Threshold analysis table

用户并发申请数量	α（平均响应指数）	β（最低响应指数）
40	0.232	0.104
200	0.229	0.103
500	0.234	0.105
800	0.232	0.105
1000	0.230	0.102

2.3 衡量指标

在 FPGA 云平台环境中，为了进一步分析用户高并发申请节点时，系统的整体性能以及用户申请节点情况，引入下列参数作为衡量指标。

用户请求响应时间 T_u 计算公式如式（2）所示，其中 T_{cs} 为用户从客户端向服务器端发出请求的时间，T_s 为服务器接受请求并处理所需的时间，T_{sc} 为服务器端返回数据给客户端的时间，T_c 为客户端接收响应数据并处理所需的时间。

$$T_u = T_{cs} + T_s + T_{sc} + T_c \tag{2}$$

吞吐量 Q_t 指单位时间内系统处理请求的数量，其计算公式如式（3）所示，S_{num} 为样本总数量，T_{last} 为最后一个线程启动的时间，$T_{lasttime}$ 为最后一个线程持续的时间，T_{first} 为第一个线程启动的时间。

$$Q_t = S_{num} / (T_{last} + T_{lasttime} - T_{first}) \tag{3}$$

平均响应延时 L_a 计算公式如（4）所示，l_i 为单个请求的响应延时。

$$L_a = \sum_i l_i / S_{sum} \tag{4}$$

并发度 C 计算公式如 (5) 所示，对于一个系统而言，平均响应延时为每个请求所需的时间，吞吐量为单位时间的请求数量，因此二者相乘即为并发度。

$$C = Q_t * L_a \tag{5}$$

根据上述公式最终可以计算得出用户请求响应时间以及微服务的并发度，从而衡量 FPGA 云平台中节点资源管理微服务的服务质量。从用户请求响应时间角度分析，响应时间越小，用户体验越好。从系统的并发度角度分析，并发度越高，系统性能越好[23]。

2.4 算法设计

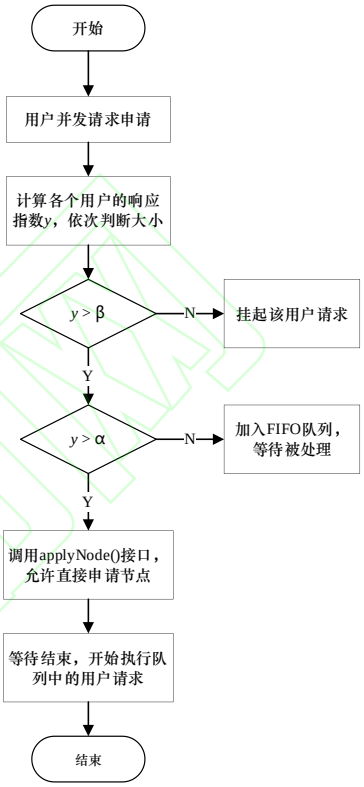

图 4　高并发算法执行流程图

Fig.4　High concurrency algorithm execution flow chart

本文在上述自定义参数响应指数计算模型的基础上，提出一种高并发请求（HCRS）调度算法，根据上述定义的阈值将用户响应指数划分为三种执行类型：直接申请、加入队列等待申请、挂起申请，并以此为标准分配用户申请节点请求。

算法的执行流程如图 4 所示，大量用户同时发起申请节点请求，输入对应的参数，如计划使用时间。传递输入参数后，模型开始计算各个用户对应的响应指数，并与阈值比较判断大小，从而分类处理不同用户请求。

基于自定义参数响应指数计算模型分析，本文设计的高并发请求调度算法具体如下：

算法 2 高并发请求调度算法

输入 用户请求参数

输出 用户请求执行顺序

```
1. array2 ← normalize(array);
2. y ← count(array2,weight); //计算响应指数
3. process_request(); //处理用户请求
4. for i in range(0,num){
5. if res < min
6. else min < res <= avg
7. q.put(res);
8. else res > avg
9. apply();}
10. process_queue();
11. for i in range(0, qsize){
12. res = q.get();
13. apply();}
```

3 实验与分析

本文将高并发请求（HCRS）调度算法应用到真实的 FPGA 云平台上，并通过在真实应用场景下模拟同一组用户高并发登录及申请节点请求，分析评估资源分配情况以及微服务的质量。

3.1 实验环境及工具

本文部署节点资源管理微服务的软件环境为 CentOS Linux release 7.8.2003(Core)+GCC V4.8.5，其底层框架为 Tornado5.1，FPGA 云平台使用的硬件节点为 Xilinx 公司发布的 PYNQ-Z2，节点系统为 PYNQ Linux, based on Ubuntu 18.04+GCC V7.3.0。Web 应用技术栈为：Spring Boot、Apache Shiro、MyBatis、Thymeleaf。数据库采用 Mysql 5.5 与 Redis 3.0 相结合的方式存储管理数据。

测试部分首先使用可视化测试工具 Badboy 将单个登录以及申请节点的测试脚本直接导出生成 Apache Jmeter 脚本，再使用 Apache Jmeter 根据具体测试要求修改测试脚本，添加用户登录信息以及用户计划使用时间即输入参数进行测试。

3.2 实验结果

本文算法已在真实的 FPGA 云平台 OpenHEC 中实现，通过模拟 1000 个用户分别在先来先服务（FCFS）调度算法以及高并发请求(HCRS)调度算法下执行登录及申请节点请求，同时还需考虑高并发请求调度算法中自定义参数的输入，为保证实验过程更接近真实用户操作，所有参数取值如下：（1）“用户历史平均使用时间”权重较高，对实验影响较大，故需获取其真实数据；（2）“用户计划使用时间”具有较大的随机性，故随机生成 1000 个用户的计划使用时间存入文本文件作为测试数据；（3）“是否是会员、用户计划申请节点数量、用户历史空置时间”这三个参数均设为默认值，默认用户均为普通用户，并且一次申请一个节点使用，默认用户历史空置时间设为零。根据以上参数的取值设定，在此基础上进行模拟测试实验。

不同算法下，1000 个用户登录以及申请节点请求通过率为 100%，请求响应时间如表 5 所示，与 FCFS 算法相比，HCRS 算法申请节点的平均响应时间降低了 12.605s。

表 5 用户请求响应时间统计表

Table 5 Statistics of user request response time

Requests	Response Times(ms)				
Label	Average	Min	Max	Median	99th pct
Apply(FCFS)	85325.60	3688	128169	86426.00	88839.89
Apply(HCRS)	72719.85	6261	106412	65285.50	98015.24
Login(FCFS)	2068.68	683	3203	2098.50	3092.97
Login(HCRS)	5214.14	1483	7023	5595.00	6937.98

两种算法在吞吐量及网络接收发送数据量如表 6 所示，申请节点请求下，HCRS 算法的吞吐量以及网络收发数据量均优于 FCFS 算法。

表 6 吞吐量及网络接收发送数据表

Table 6 Throughput and network receiving and sending data table

Requests	Throughput	Network(KB/sec)	
Label	Transactions/s	Received	Sent
Apply(FCFS)	7.80	2.33	3.92
Apply(HCRS)	9.35	2.45	4.73
Login(FCFS)	304.32	123.65	129.02
Login(HCRS)	141.70	57.57	59.66

衡量微服务质量的另一指标为用户请求平均响

应延时，1000 个用户并发请求执行完毕所需总时长为两分钟。图 5 对比显示了两种算法在总时长内每隔一分钟的平均响应延时，从图中可以得知，HCRS 算法的请求平均响应延时略高于 FCFS 算法。

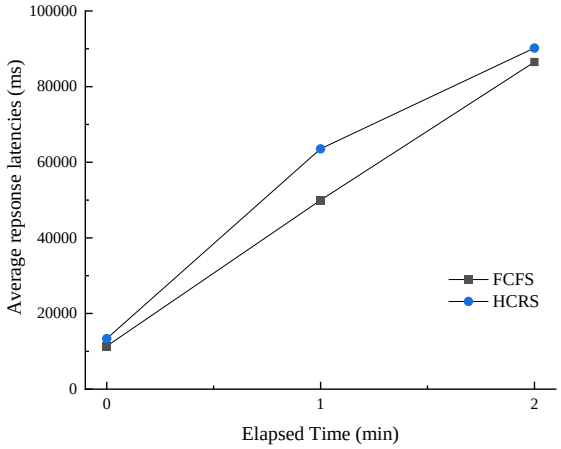

图 5　不同算法平均响应延时对比图

Fig.5　Comparison of average response latencies of different algorithms

3.3 实验结果分析

3.3.1 用户请求响应时间分析

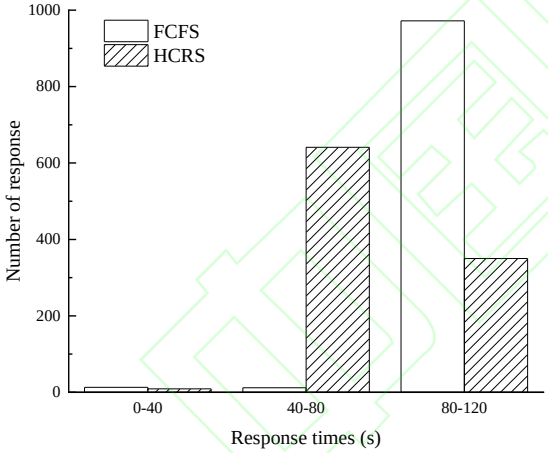

图 6　不同响应时间区间内处理请求统计图

Fig.6　Statistical of processing requests in different response time intervals

上述实验结果表明，HCRS 算法申请节点请求平均响应时间优于 FCFS 算法。从新的维度分析，即两种算法在不同响应时间区间内处理请求个数，如图 6 所示，HCRS 算法大部分请求响应时间处于 40-80 秒之间，少部分请求处于 80-120 秒之间，而 FCFS 算法几乎所有请求的响应时间都处于 80-120 秒之间。

因此，两种算法对比分析可知，在处理用户并发申请节点请求上，HCRS 算法能有效降低请求响应时间。

3.3.2 系统吞吐量分析

从实验结果分析可知，HCRS 算法的系统吞吐量相较于 FCFS 算法更优。同时从图 7 各时刻系统吞吐量分析可知，HCRS 算法更为稳定，前期处于上升趋势，中期最高达到每秒处理 11 个请求，后期回落到每秒处理 6 个请求，而 FCFS 算法的系统吞吐量则到后半程才逐渐上升，最高达到每秒处理约 16 个请求，相比之下，使用高并发请求调度算法的系统吞吐量更为稳定，并且性能更优。

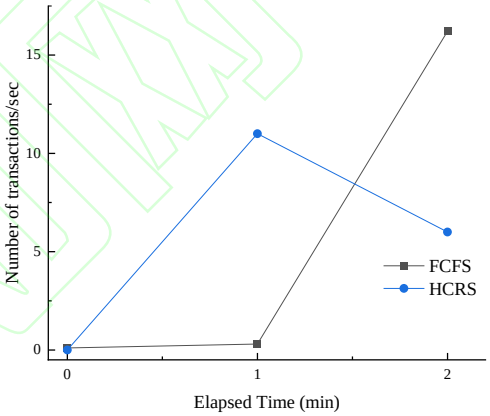

图 7　不同算法各时刻系统吞吐量对比图

Fig.7　Comparison of system throughput at of different algorithms at different times

3.3.3 用户请求平均响应延时分析

由实验结果可知，HCRS 算法下用户请求平均响应延时要高于 FCFS 算法，相比之下并没有有效降低响应延时。但从完成一个完整的请求所需时间即响应延时与每秒请求数关系分析来看，如图 8 和 9 所示，并发处理用户请求的数量集中在 250 个请求以内，对比分析这段区间内的响应延时，FCFS 算法的响应延时多数分布在 80-100 秒之间，少数分布在 0-50 秒之间，最高达到 128 秒左右。而 HCRS 算法的响应延时多数分布在 50-100 秒之间，少数分布在 0-20 秒之间，最高达到 95 秒。在上述情况下，HCRS 算法处理请求延时普遍低于 FCFS 算法。

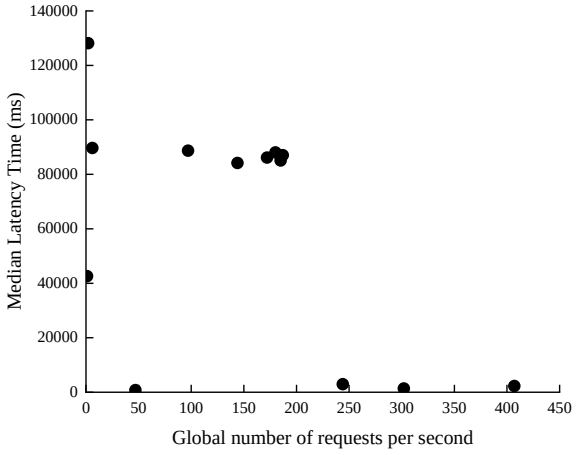

图 8 FCFS 平均响应延时与每秒请求数关系图

Fig.8 Relationship between FCFS average latency time and requests per second

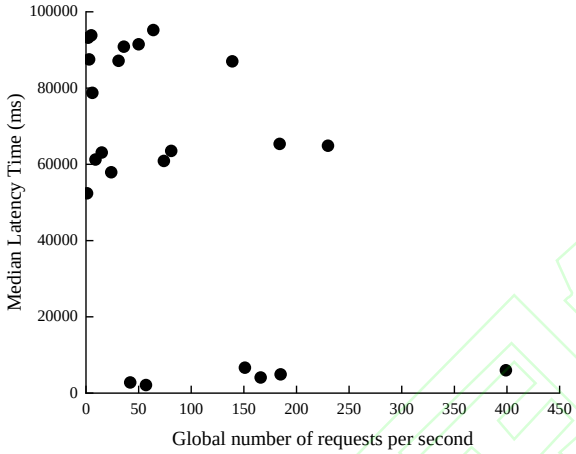

图 9 HCRS 平均响应延时与每秒请求数关系图

Fig.9 Relationship between HCRS average latency time and requests per second

分析对比两种算法每秒处理相同数量用户请求的响应延时可知，如表 7 所示，HCRS 算法平均响应延时降低 29.074s。

表 7 每秒处理相同数量请求响应延时统计表

Table 7 Response latency statistics table for processing the same number of requests per second

Algorithm	response Latency（ms）			
Requests per second	1	2	6	185
FCFS	42605	128164	89673	85078
HCRS	52400	93199	78738	4887

3.3.4 并发度分析

由上述衡量指标中并发度公式可以计算出 FCFS 算法以及 HCRS 算法的并发度，两种算法的吞吐量以及各时刻平均响应延时如表 8 所示。

表 8 吞吐量及各时刻平均响应延时数据表

Table 8 Data table of throughput and average response latency at each time

Algorithm	Average response Latency（ms）			Throughput
Name	0	1	2	Transactions/s
FCFS	11358	49962	86507	7.80
HCRS	13342	63557	90216	9.35

根据表 8 数据计算在 0,1,2 时刻下 FCFS 算法的并发度分别为：

$C_0 = 88.5924$，$C_1 = 389.7036$，$C_2 = 674.7546$；

而 HCRS 算法的并发度为：

$C_0 = 124.7477$，$C_1 = 594.2580$，$C_2 = 843.5196$；

两种算法总并发度为各时刻并发度之和，具体如下：

$C_{FCFS} = 1153.0506$，$C_{HCRS} = 1562.5253$。

分析比较两种算法各时刻并发度以及总并发度，高并发请求（HCRS）调度算法的并发度均要高于先来先服务（FCFS）调度算法，因此，在加入了高并发调度算法后，能有效提升系统并发度，同时提升微服务质量。

3.4 实验总结

本文分别在应用了不同算法的真实 FPGA 云平台上进行实验，模拟同一组数量为 1000 的用户进行登录及申请节点操作，从而得到实际应用场景中的用户请求测试实验结果。

对比分析两种算法的测试实验结果，在处理用户并发申请节点请求时，HCRS 算法能使请求响应时间降低 12.605s；并且在提高系统吞吐量以及网络接收、发送数据量的同时使得系统更为稳定；在每秒处理相同数量的用户请求的情况下，平均响应延时降低 29.074s。最终提高系统并发度，提升微服务质量。

上述测试实验结果表明，高并发请求调度（HCRS）算法能在满足用户高并发申请节点的情况下，优化系统性能，合理分配硬件节点资源，提升硬件节点资源利用率。

4 结束语

本文针对微服务架构下的 FPGA 云平台用户请求并发控制问题，提出了一种高并发请求（HCRS）调度算法。结合自定义参数响应指数计算模型，计算各个用户的响应指数，根据响应指数阈值进行划分类别，采取了优先级调度的模式处理用户申请节点请求。

在高并发请求场景下，该算法能够有效降低平均响应延时、请求响应时间，提升系统吞吐量以及网络接收、发送数据量，并且提高系统并发度，优化系统性能。与此同时，能够满足历史行为较为友好以及使用频率较高的用户优先使用节点，提升硬件节点资源利用率。目前，改进后的 FPGA 云平台已上线投入使用，有效解决用户并发申请节点问题，取得较好效果。

为了进一步优化云平台性能及微服务质量，未来将针对系统容错、网络通信安全及服务监控等方面进行研究，并对微服务架构中负载均衡机制进行优化改进，通过不同的高并发原则来提升系统吞吐量。

参考文献

[1] Alves G, Fidalgo A, Marques M, et al. Spreading remote lab usage: A system — A community — A Federation[C]//Engineering Education. Vila Real, Portugal: IEEE Press, 2016: 1-7.

[2] Syed A A, Asif R, Hina S, et al. Cloud Based Remote FPGA Lab Platform: An Application of Internet of Things[J]. Mehran University Research Journal of Engineering and Technology, 2018, 37(4): 535-544.

[3] Doshi J, Patil P, Dave Z, et al. Implementing a cloud based Xilinx ISE FPGA design platform for integrated remote labs[C]//International Conference on Advances in Computing. Kochi, India: IEEE Press, 2015.

[4] Fujii N, Koike N. IoT Remote Group Experiments in the Cyber Laboratory: A FPGA-based Remote Laboratory in the Hybrid[C]//International Conference on Cyberworlds. Chester, UK: IEEE Press, 2017.

[5] Toyoda Y, Koike N, Li Y. An FPGA-based remote laboratory: Implementing semi-automatic experiments in the hybrid cloud[C]//International Conference on Remote Engineering & Virtual Instrumentation. Madrid, Spain: IEEE Press, 2016.

[6] 施凌鹏, 朱征, 周俊松, 李鑫, 李静. 面向微服务架构的云系统负载均衡机制[J]. 计算机工程, 2021, 47(9): 44-50,58.
SHI Lingpeng, ZHU Zheng, ZHOU Junsong, LI Xin, LI Jing. Load Balancing Mechanism for Microservice Architecture in Cloud-based Systems[J]. Computer Engineering, 2021, 47(9): 44-50,58.

[7] Fowler M, Lewis J. Microservices[EB/OL]. [2021-8-20]. https://martinfowler.com.

[8] Kainz A. Microservices vs. Monoliths: An Operational Comparison[EB/OL]. [2020-7-7]. https://thenewstack.io/microservices-vs-monoliths-an-operational-comparison.

[9] Liu X, Zhai X, Lu W, et al. QoS-guarantee Resource Allocation for Multibeam Satellite Industrial Internet of Things with NOMA[J]. IEEE Transactions on Industrial Informatics, 2019, PP(99): 1-1.

[10] Shetty J, D'mello D A. QoS Based Cloud Service Selection to Handle Large Volume of Concurrent Requests[J]. Indian Journal of Science and Technology, 2016, 9(48).

[11] Sun M, Zhou Z, Wang J, et al. Energy-Efficient IoT Service Composition for Concurrent Timed Applications[J]. Future Generation Computer Systems, 2019, 100.

[12] GUO Jun, WU Jing, XING Liu-Dong, et al. A Coping Strategy for Bursty Workload of Cloud Service[J]. Chinese Journal of Computers, 2019, 42(4): 864-882. (in Chinese)
郭军, 武静, 邢留冬, 等. 面向突发业务的云服务并发量应对策略研究[J]. 计算机学报, 2019, 42(4):

864-882.

[13] Al A M a B A, Mamoona H, Nz J. Proposing a Load Balancing Algorithm For Cloud Computing Applications[J]. Journal of Physics: Conference Series, 2021, 1979(1).

[14] Wei L, Dongwei Z, Zhijun G. A novel load balancing strategy based on node load comprehensive measuring under cloud computing environment[J]. Journal of Computational Methods in Sciences and Engineering, 2019, 19(S1).

[15] Zhou J, Cen B, Cai Z, et al. Workload Modeling for Microservice-Based Edge Computing in Power Internet of Things[J]. IEEE Access, 2021, 9: 76205-76212.

[16] Al-Debagy O, Martinek P. A Comparative Review of Microservices and Monolithic Architectures[C]// International Symposium on Computational Intelligence and Informatics. Budapest, Hungary: IEEE Press, 2018.

[17] Villamizar M, Garcés O, Ochoa L, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures[J]. Service Oriented Computing & Applications, 2017, 11(6): 1-15.

[18] Lu X, Phang K. An enhanced hadoop heartbeat mechanism for MapReduce task scheduler using dynamic calibration[J]. China Communications, 2018, 15(11): 93-110.

[19] Rochim A F, Aziz M A, Fauzi A. Design Log Management System of Computer Network Devices Infrastructures Based on ELK Stack[C]//2019 International Conference on Electrical Engineering and Computer Science (ICECOS). 2019: 338-342.

[20] DENG Xue, LI Jiaming, ZENG Haojian, et al. Research on Computation Methods of AHP Wight Vector and Its Applications[J]. Mathematics in Practice and Theory, 2012, 42(7): 93-100. (in Chinese)

邓雪, 李家铭, 曾浩健, 等. 层次分析法权重计算方法分析及其应用研究[J]. 数学的实践与认识, 2012, 42(7): 93-100.

[21] LIU Wei-Feng, HE Xia. Two-step Method of Group Decision Making Based on Fuzzy Judgment Matrix[J]. Computer Engineering, 2012, 38(10): 141-143. (in Chinese)

刘卫锋, 何霞. 基于模糊判断矩阵的两阶段群决策方法[J]. 计算机工程, 2012, 38(10): 141-143.

[22] Rudrabhatla C K. A Quantitative Approach for Estimating the Scaling Thresholds and Step Policies in a Distributed Microservice Architecture[J] IEEE Access, 2020, 8: 180246-180254.

[23] Bae J, Jang H, Jin W, et al. Jointly optimizing task granularity and concurrency for in-memory mapreduce frameworks[C]// 2017 IEEE International Conference on Big Data (Big Data). 2017: 130-140.