计算机科学
COMPUTER SCIENCE

面向利润优化的软实时云服务调度与多服务器系统配置方法研究

王添, 沈伟, 张功萱, 徐林丽, 王震, 郁云

引用本文

王添, 沈伟, 张功萱, 徐林丽, 王震, 郁云. 面向利润优化的软实时云服务调度与多服务器系统配置方法研究[J]. 计算机科学, 2024, 51(6A): 230900099-10.

WANG Tian, SHEN Wei, ZHANG Gongxuan, XU Linli, WANG Zhen, YUN Yu. Soft Real-time Cloud Service Request Scheduling and Multiserver System Configuration for ProfitOptimization [J]. Computer Science, 2024, 51(6A): 230900099-10.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230900099

面向利润优化的软实时云服务调度与多服务器系统配置方法研究

王 添[1] 沈 伟[4] 张功萱[2] 徐林丽[3] 王 震[1] 郁 云[1]
1 江苏财会职业学院信息工程学院 江苏 连云港 222061
2 南京理工大学计算机科学与工程学院 南京 210094
3 江苏海洋大学理学院 江苏 连云港 222061
4 阿里云(中国) 杭州 310030

摘 要 在云计算中,由于多核技术的不断革新,近年来有许多工作研究了基于多核处理器的多服务器系统。云服务提供商通过建立多服务器系统给用户提供云服务并优化云服务利润是目前云计算领域的一个热点问题,对这些问题的研究推动着云计算技术的不断发展。然而,现有的关于多服务器系统的研究要么局限于通过对多服务器计算资源的配置来优化云服务利润而忽视了云服务请求本身的可调度性,要么局限于开发服务请求调度策略来提升云服务利润而忽视了多服务器系统的动态扩展性。但若使用云服务请求调度与多服务器配置协同优化来提升云服务利润,则会使问题规模的复杂性呈指数增长。因此,为云服务提供商设计一个面向软实时云服务请求的云服务调度与多服务器配置方法是十分必要的。此外,现有的研究在配置多服务器系统时大多忽略了处理云服务请求会遭受瞬时故障的情况。而许多研究表明,软实时任务在遭受瞬时故障时会影响服务请求的执行结果从而影响云服务利润。本研究面向软实时云服务请求,针对云环境中普遍存在的计算性能异构的服务器资源,开发了一个基于深度搜索的灰狼算法来协同优化云服务请求调度和多服务器配置以最大化云服务利润。最后,为了验证所提方法的有效性,进行了大量实验,实证结果表明,与现有的基准方法相比,所提方法得到的云服务利润平均增加了 6.83%。

关键词:云计算;多服务器系统配置;云服务请求调度;云服务利润最大化;软错误可靠性
中图分类号 TP391

Soft Real-time Cloud Service Request Scheduling and Multiserver System Configuration for Profit Optimization

WANG Tian[1],SHEN Wei[4],ZHANG Gongxuan[2],XU Linli[3],WANG Zhen[1] and YUN Yu[1]
1 School of Information Science,Jiangsu College of Finance and Accounting,Lianyungang,Jiangsu 222061,China
2 School of Computer Science and Engineering,Nanjing University of Science and Technology,Nanjing 210094,China
3 School of Science,Jiangsu Ocean University,Lianyungang,Jiangsu 222061,China
4 Alibaba Cloud,Hangzhou 310030,China

Abstract In cloud computing,there has been considerable research on multi-server systems based on the continuous innovation of multi-core technology. Establishing multi-server systems to provide cloud services to users and optimizing cloud service profitability is a hot topic in cloud computing. Research on these issues drives the continuous development of cloud computing technology. However,existing studies on multi-server systems either focus on optimizing cloud service profitability through the configuration of multi-server computing resources,neglecting the schedulability of cloud service requests themselves,or concentrate on developing service request scheduling strategies to improve cloud service profitability while overlooking the dynamic scalability of multi-server systems. However,when using coordinated optimization of cloud service scheduling and multi-server configuration to enhance cloud service profitability,the complexity of the problem increases exponentially. Therefore,it is essential to design a cloud service scheduling and multi-server configuration method for providers targeting soft real-time cloud service requests. Besides,existing research on configuring multi-server systems often overlooks the transient faults in processing cloud service requests. Numerous studies have demonstrated that soft real-time tasks can be affected by transient faults,leading to variations in the execution results of service requests and impacting cloud service profitability. In this study,we focus on soft real-time cloud service requests and develop a depth-search-based grey wolf algorithm to jointly optimize cloud service request scheduling and

基金项目:国家自然科学基金(62272232);教育部产学研创新基金(2022IT224);江苏高校哲学社会科学研究项目(2023SJYB1851);连云港市基础研究计划基金(JCYJ2328);江苏财会职业学院科研启动项目(2023GC06);江苏财会职业学院基础科研项目(2023XJ19)
This work was supported by the National Natural Science Foundation of China(62272232),Industry University Research Innovation Foundation of the Ministry of Education of China(2022IT224),College Philosophy and Social Science Foundation of Jiangsu(2023SJYB1851),Lianyungang Research Foundation for Basic Research(JCYJ2328),Science Research Staring Foundation of Jiangsu College of Finance and Accounting(2023GC06) and Basic Research Foundation of Jiangsu College of Finance and Accounting(2023XJ19).
通信作者:王震(wangzhen@jscfa.edu.cn)

multi-server configuration, considering the prevalent computational performance heterogeneity of server resources in cloud environments, aiming to maximize cloud service profitability. Finally, extensive experiments validate the effectiveness of the proposed method. The empirical results demonstrate that, compared with the existing benchmark methods, the cloud service profits obtained by the proposed method increase by an average of 6.83%.

Keywords Cloud computing, Multi-server system configuration, Cloud service request scheduling, Cloud service profit maximization, Soft error reliability

1 引言

云计算是以共享的方式将物理资源虚拟化为通用资源池，并使其产品化的大规模分布式并行计算系统[1-5]。云计算拥有强大的基础设施资源和服务整合能力。云用户可通过互联网访问云计算系统，按需购买并接收所需的硬件资源和服务。云计算通过封装底层硬件资源，将相应的上层服务转化为按需付费的商品后，云服务提供商将不再需要专注于硬件资源的采购和维护，从而可以专注于自己的云服务业务。与所有商业服务一样，提高云服务的盈利能力是云服务提供商们共同追逐的目标。此外，云服务利润也是推动云计算技术不断发展的潜在驱动力。因此，对云服务提供商来说，理解云计算的经济性(例如，多服务器配置/云服务请求负载与云服务收入/成本之间的关系)就显得非常重要。云计算系统又称多服务器系统[1-5]，其物理资源通常包含计算单元(如CPU，GPU)、存储资源和网络资源。而多服务器系统性能主要依赖于为其配置的计算单元的性能，其他的资源在配置时可以根据计算单元性能按比例缩放[6]。目前已有越来越多的云服务提供商主要依赖配置的计算资源来对用户收取相应的费用[1-5,7]。因此，本文中的多服务器系统配置主要关注系统的计算资源配置策略研究。

云服务提供商的利润来自于向云客户提供云服务所获得的服务收入与处理服务请求所消耗的资源成本之间的差额。增加收入或降低成本都可以提高云服务提供商的利润。由文献[1-5]可知，在云服务市场中，云服务收入与服务水平协议(Service Level Agreement，SLA)密切相关。尽管服务水平协议中有许多不同的服务性能指标[8]，但云计算系统中广泛使用的性能指标是服务响应时间(也称为周转时间)[3-4,7,9]，这是完成一个云服务请求所花费的总时间，即系统中云服务等待时间和云服务执行时间之和。云服务提供商的成本主要来自构建多服务器系统的资源成本和能耗成本。已经有许多研究工作[3,4,7,9-13]致力于通过优化资源管理(如多服务器配置)来提高云服务利润，另外一些[14-17]通过优化服务请求调度(如服务请求分发策略)来提高云服务利润，但只有很少部分人考虑到了服务器的计算性能异构的问题。此外，在基于COMS技术的计算单元中运行云服务请求时，服务请求将不可避免地遭受瞬时故障的干扰。瞬时故障是由硅基芯片依赖的半导体制造工艺导致的。随着半导体制造工艺不断进步，芯片中的晶体管数目和集成度不断增加，从而导致芯片的电路系统极易受到电磁干扰、瞬时电压扰动、高能粒子撞击的影响(即瞬时故障)。由于瞬时故障只会妨碍云服务请求的执行并不会对物理硬件造成损伤，因此瞬时故障又称为软错误。如果用户提交的云服务请求因软错误而未能正确执行，云服务收益必然会受到影响，这对云服务提供商来说显然是不可接受的。因此，本文在配置多服务器系统时将充分考虑软错误的影响，并基于计算性能的异构特性，提出多服务器系统配置和云服务请求调度协同优化方法，可以在保证云用户的服务水平协议的同时，最大化云服务提供商的利润。

本文的研究工作基于一个云计算应用场景：云服务提供商构建云服务平台，为用户提供多种类型的云服务请求处理服务，根据云服务请求的类型配置特定的多服务器系统。该云平台包含多个计算性能异构的多服务器系统，每个多服务器系统各自拥有限个计算性能同构的计算单元可供扩展。本文的主要贡献总结如下：

1)在云服务三层架构中，本工作为云服务提供商设计了一个资源配置与云服务请求调度方法，该方法充分考虑了计算资源的异构性与云服务请求的多样性，可充分利用性能异构的服务器资源。

2)弥补了现有云服务的服务水平协议在计算云服务响应时间时未考虑多服务器系统规模的缺陷，并考虑了云服务请求在执行时遭受软错误的情况，设计了软错误容错机制，并与基于软错误可靠的多服务器系统配置问题进行了有效的统一，形式化为一个云服务利润最大化问题。

3)为了解决面向云服务利润优化的多服务器系统配置及服务请求调度问题，设计了一个基于深度搜索的灰狼算法以最大化云服务提供商的利润。与基准方法相比，我们的方法可以平均提升6.83%的云服务利润。

本文第2章回顾了云服务利润最大化的相关工作；第3章介绍了本文涉及的云计算系统模型，包括云服务三层结构模型、云服务平台模型、云服务水平协议模型、云服务成本模型、软错误可靠性模型及多服务器系统配置与任务调度问题模型；第4章详细介绍了基于灰狼算法和深度搜索算法设计的多服务器配置和云服务请求调度方法；第5章通过大量的实验来评估本文方法的有效性；最后总结全文。

2 相关工作

对于云服务提供商来说，要想实现云服务利润的最大化，通常可以通过增加服务收益或是降低云服务处理成本来实现。本章将从计算资源的管理和云服务请求的调度两个方面介绍相关研究工作。

在云数据中心中，智能资源管理策略提高了资源利用率，降低了能源消耗，并使云服务提供商受益。为了使服务提供商的利润最大化，在资源管理方面已经进行许多研究工作。Cong等[3]提出了一种基于客户感知价值的利润最大化方案，该方案反映了服务请求与顾客服务提供商之间的动态交互。针对云服务市场需求的不确定性及云服务合同中蕴含的不确定性风险，Wang等[4]提出了一种基于客户感知价值和风险意识的多服务器配置，以最大限度地提高服务提供商的利润。Cao等[7]提出了一种基于动态电压缩放(Dynamic Voltage Scaling，DVS)的方法来优化云数据中心服务器的运行速度，

以达到节省能耗成本、提升云服务利润的目标。Goudarzi等[10]提出了一种基于SLA的云服务利润最大化启发式算法，该算法通过关闭不必要的服务器来优化多服务器系统。Mei等[11]提出了一种云服务器的混合租赁策略，利用服务器的长期和短期租赁机制开发了针对不同云服务需求的多服务器计算资源的配置策略。在Mei等工作的基础上，Kang等[12]开发了一种基于利润驱动的资源管理方法，该方法的主要贡献是将云计算资源划分为更细粒度的定价类别。Mei等[9]还提出了一种基于云用户满意度的云服务利润最大化方案，该方案考虑了客户满意度对云服务需求和多服务器配置的影响。Xu等[13]也提出了一种通过预测当前云服务需求来匹配云计算资源使用状态的云服务动态定价策略，以确保云服务提供商的云服务利润。尽管上述研究工作从不同的角度探讨了如何利用云计算资源管理策略来最大化云服务提供商的利润，但要么忽略了云计算环境中计算资源性能异构的问题，要么忽略了云服务请求调度优化问题。

还有一些研究对如何优化服务请求调度以降低多服务器系统的运营成本和提高云服务提供商利润做出了贡献。Ali等[14]根据任务的属性将任务划分为不同的类别，并根据任务服务质量的要求，为不同类型的任务设置不同的优先级，从而制定了任务调度策略。Chen等[15]开发了一种任务调度策略，在满足成本约束的同时，最大限度地减少云服务的调度时长。该策略是通过提高资源利用率来降低云数据中心的运营成本。在大多数情况下，云数据中心的任务调度是一个NP难问题，很难在可行的时间内得出最优解。因此，Tian等[16]提出了一种基于粒子群优化(PSO)的任务调度方法，该方法利用粒子的适应度来寻找次优可行解。与Tian等类似，Deng等[17]提出一种新的改进的粒子群优化算法对云数据中心的任务进行调度优化，该算法中子种群之间的局部和全局进行信息交换，并在迭代过程中推动解向最优方向移动。上述工作虽然针对不同的场景提出了多种任务调度的方法，最大限度地提高云服务提供商的效益，但忽略了对云数据中心计算资源差异的协同优化。此外，无论是针对计算资源管理或是针对服务请求调度的云服务利润最大化方法，大多都没有考虑任务执行过程中的瞬时故障发生率会影响任务执行的问题。

3　系统模型

本章将介绍云计算的系统模型，包括云服务三层结构模型、由多服务器系统构成的云服务平台模型、基于弹性云的服务水平协议模型、云服务处理运行的成本模型及多服务器配置与任务调度协同优化问题的模型。

3.1　云服务三层结构模型

本文考虑了一个三层云服务模型[12-13](见图1)，该模型具有3个典型的实体，即云基础设施提供商(Cloud Infrastructure Provider，CIP)、云服务提供商(Cloud Service Provider，CSP)和云服务用户(Cloud Customer，CC)。在公有云计算环境中，CIP为IaaS(基础设施即服务)提供商，CSP为SaaS/PaaS(软件即服务/平台即服务)提供商。

在云服务三层结构中，CIP拥有硬件设施资源，CSP向CIP租赁云计算基础设施资源，构建多服务器系统，构建的多服务器系统以虚拟机(VM)的形式向CC提供云服务(如科学计算、电子商务数据计算、图形图像处理，如图1所示)并收取相应的费用。CC向CSP提交云服务请求，并根据服务水平协议商定的结果支付相应的费用。

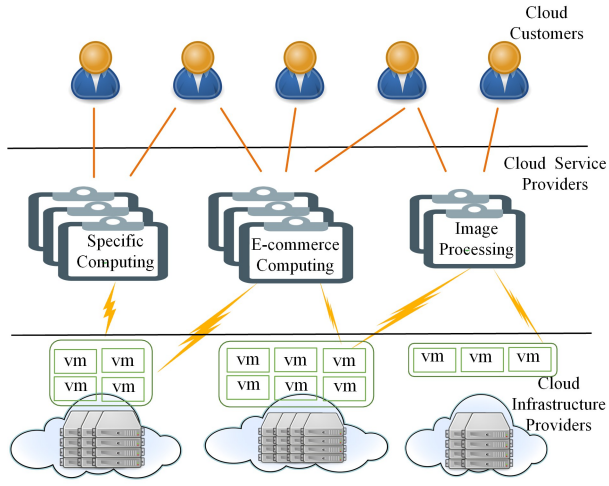

图1　三层云服务模型[19]

Fig. 1　Three-tier cloud service model[19]

根据文献[1-5]所述，CSP可在概念上与进行CIP区分，因为CSP将探索优化多服务器配置的智能算法，以处理CC提交的服务请求并返回CC所需要的结果。换言之，CSP优化了多服务器系统配置，以追求云服务效益最大化，而CIP只是根据租用的基础设施资源向CSP收费。但在实践中，CSP也可以是CIP(即CSP可同时提供IaaS，PaaS和SaaS云服务)，而我们的三层模型即可以通过组合租赁成本和公用事业成本来覆盖实践中的这种情况，也可以设立一个单独的CSP，只为CC提供PaaS或SaaS服务来从概念上区分CIP。这种具有吸引力的三层体系结构常见于具有三方的集群计算系统(即集群节点、集群管理器和消费者)，或具有三方的网格计算系统(即资源提供商、服务提供商和客户端)[5]。这种三层云服务模式是当前云服务计算领域的研究热点，且被实际的云服务市场广泛关注。以Apple中国为例，向云上贵州公司租赁基础设施资源为客户提供iCloud云服务并遵守中国的法律[18]。

3.2　云服务平台模型

本文的云服务平台由M个不同的多服务器系统构成，多服务器系统结构可以是多种形态的[3-4]，例如多服务器系统可以是一组刀片服务器、一组商用服务器或是一个多核处理器，且其中刀片、处理器、内核可统称为服务器。在本研究中，我们构建的是多核处理器级的多服务器系统，一个服务器即一个内核(或一个算计单元)。每个多服务器系统中含有一组性能相同的内核，且被部署在特定的运行环境以满足云服务应用域的运行要求。任意一个多服务器系统可表示为MS_j($j=1,2,3,\cdots,M$)，即云平台中的j型多服务器系统，多服务器系统中的服务器运行速度可以从预先定义的速度集中进行设置，服务器类型按照预先定义的速度集异同进行分类。每种类型的服务器拥有有限的数量，不同类型的服务器租赁价格可能不同。云服务平台可以服务多种类型的应用程序域(Application Domain，AD)，用AD_i表示第i个应用程序域。此外，云服务平台拥有一个全局调度器(Scheduler)，如图2所示。全局调度器可以周期性预测各AD的云服务请求到达数

量，即云服务的平均到达率(λ)。当云服务请求到达时，全局调度器使用特定的工作负载分配策略将服务请求立即调度到合适的 MS。在本文中，我们假设到达云服务平台的服务请求是一个泊松流，且每个 MS 使用具有一定容量的等待队列来容纳从 Scheduler 调度来的服务请求，并按“先到先服务”(First Come First Served，FCFS)的原则执行云服务请求。

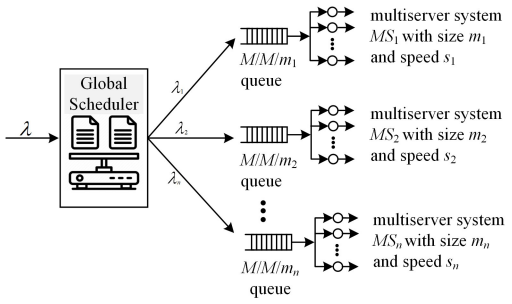

图 2 云服务平台架构[20]

Fig. 2 Architecture of cloud service platform[20]

为了对云服务平台的性能进行量化分析，需要为其构建形式化模型，因此我们基于上述对多服务器系统的合理假设，将任意一个多服务器系统 MS_j 建模为一个 $M/M/M_j$ 的排队系统。在该系统中，有 m_j 个服务器可以以相同的速度 s_j 运行，其中 s_j 可以从一组预设的速度集中进行选择，即动态电压缩放(DVS)。多服务器系统执行用户提交的云服务请求所需消耗的计算资源可以被云服务请求的执行指令数 r 量化(特别地，执行指令数 r_i 表示应用程序域 AD_i 的云服务请求的执行指令数)。因此，当服务器以速度 s 运行时，该服务器执行一个云服务请求所需的时间(t)可计算为：$t=r/s$。由文献[3-4,7]可知，任意服务请求的执行指令数 r 满足指数分布，可设其均值为 $\bar{r}$，则 r 的概率分布函数可表示为：$f_r(z)=1/\bar{r}\cdot e^{-z/\bar{r}}$，其中 z 为 r 所有可能的取值。因此，运行时间 t 也服从指数分布，且 $\bar{t}=\bar{r}/s$。令 μ 表示任意服务器的服务率(即在单位时间内，服务器能够以速度 s 处理服务请求的平均数量)，则 $\mu=1/\bar{t}=s/\bar{r}$。相应地，该服务器所属多服务器系统 MS 的系统服务率(ρ)可由式(1)计算：

$$\rho=\lambda/(m\mu)=\lambda\bar{r}/(ms) \tag{1}$$

特殊地，令 P_k^j 表示 $M/M/M_j$ 排队系统中有 k 个云服务请求正在处理或正在等待处理的概率，则根据文献[21]，P_k^j 的计算式为：

$$P_k^j=\begin{cases}P_0^j\dfrac{(m_j\rho_j)^k}{k!}, & k\leqslant m_j\\ P_0^j\dfrac{m_j^{m_j}\rho_j^k}{m_j!}, & k>m_j\end{cases} \tag{2}$$

其中，P_0^j 表示 $M/M/M_j$ 排队系统没有任务需要执行的概率，其计算式为：

$$P_0^j=\left(\sum_{k=0}^{m_j-1}\frac{(m_j\rho_j)^k}{k!}+\frac{(m_j\rho_j)^{m_j}}{m_j!}\cdot\frac{1}{1-\rho_j}\right) \tag{3}$$

令 P_b^j 定义为一个新到的云服务请求在 $M/M/M_j$ 系统中需要等待的概率，其计算式为：

$$P_b^j=\sum_{k=m_j}^{\infty}P_k^j=\frac{P_{m_j}}{1-\rho_j} \tag{4}$$

其中，P_{m_j} 表示系统中有 m_j 个云服务请求正在被处理的概率，可以通过式(2)推导计算得出。

令 $f_{W_j}(t)$ 表示任意云服务请求被调度至 $M/M/M_j$ 系统时的等待时间概率密度函数，表示为：

$$f_{W_j}(t)=(1-P_b^j)u(t)+m_j\mu_jP_{m_j}e^{-(1-\rho_j)m_j\mu_jt} \tag{5}$$

其中，$u(t)$ 是一个脉冲函数，其形式为：$u(t)=\lim_{z\to\infty}u_z(t)$，计算式为：

$$u_z(t)=\begin{cases}z, & 0\leqslant t\leqslant\dfrac{1}{z}\\ 0, & t>\dfrac{1}{z}\end{cases} \tag{6}$$

根据文献[4]可知，$u(t)$ 的计算满足以下条件：

$$\int_0^{\infty}u_z(t)\mathrm{d}t=1 \tag{7}$$

$$\int_0^{\infty}t\cdot u_z(t)\mathrm{d}t=z\int_0^{1/z}t\mathrm{d}t=\frac{1}{2z} \tag{8}$$

3.3 服务水平协议(SLA)

SLA 是云服务用户与云服务提供商之间就云服务质量和价格进行协商，以确保云服务的服务满意度。本文采用一个广泛使用的 SLA 模型，该模型可根据云服务请求的执行指令数和响应时间(R)来定义云服务请求的服务费用，即：

$$C(r,R)=\begin{cases}ar, & \text{if } R\leqslant\dfrac{cr}{s_0}\\ ar-d\left(R-\dfrac{cr}{s_0}\right), & \text{if } \dfrac{cr}{s_0}<R<\left(\dfrac{a}{d}+\dfrac{c}{s_0}\right)r\\ 0, & \text{if } R\geqslant\left(\dfrac{a}{d}+\dfrac{c}{s_0}\right)r\end{cases} \tag{9}$$

其中，a 是每单位数量指令数执行的服务费用，d 是惩罚因子(当云服务质量低时用以补偿云服务用户)，c 是一个常数，s_0 是用户处理云服务请求的预期速度，这些参数都与 SLA 中的用户满意度相关。式(9)针对 3 种不同的云服务质量来收取云服务费用，具体来说：1)如果服务请求的响应时间 R 小于 cr/s_0，则表明云服务提供商对用户提供了高质量的云服务。此时，云服务请求的收费与云服务请求的执行指令数成正比，即 ar。2)如果一个云服务请求的响应时间 R 在区间$[cr/s_0,(a/d+c/s_0)r]$，则表明云服务提供商提供了较低质量的云服务。3)如果响应时间 R 超过 $(a/d+c/s_0)r$，则表明此时的响应时间已经超出了用户的预期，云服务提供商将为用户提供免费的执行服务。由于响应时间 R 无法直接预知，但是可以由服务等待时间和服务执行时间间接计算得出，即 $R=W+r/(ms)$，将其带入式(9)，需要注意的是，这里的云服务的相应时间 R 考虑了云服务请求在多服务器系统中运行时多服务器任何可能的配置 m 和 s，可得：

$$C(r,W)=\begin{cases}ar, & \text{if } 0\leqslant W\leqslant\left(\dfrac{c}{s_0}-\dfrac{1}{ms}\right)r\\ \left(a+\dfrac{cd}{s_0}-\dfrac{d}{ms}\right)r-dW, & \text{if } \left(\dfrac{c}{s_0}-\dfrac{1}{ms}\right)r<W\leqslant\left(\dfrac{a}{d}+\dfrac{c}{s_0}-\dfrac{1}{ms}\right)r\\ 0, & \text{if } W>\left(\dfrac{a}{d}+\dfrac{c}{s_0}-\dfrac{1}{ms}\right)r\end{cases} \tag{10}$$

特殊地，当云服务请求应用程序域 AD_i 在多服务器系统 MS_j 中处理时，根据式(10)与云服务请求执行指令数的概率密度函数 $f_{r_i}(z)$ 及等待时间的概率密度函数 $f_{W_j}(t)$，可推导出理想情况下(即云服务请求没有遭受软错误的影响时)，

处理应用程序域 AD_i 的平均收入为：

$$\bar{C}_i(\lambda_i,m_j,s_j)=a\bar{r}_i\left(1-\frac{P_b^j}{(m_js_j-\lambda_i\bar{r}_i)\left(\frac{c}{s_0}-\frac{1}{m_js_j}\right)+1}\times\frac{1}{(m_js_j-\lambda_i\bar{r}_i)\left(\frac{a}{d}+\frac{c}{s_0}-\frac{1}{m_js_j}\right)+1}\right) \tag{11}$$

3.4　云服务成本模型

在云服务市场中，CSP 对云基础设施资源的使用成本主要由两部分组成：资源使用成本（即租赁费）和运行成本（即能耗费）。服务器的租赁单价设为 β，则租赁 m 个服务器的租赁成本（$Cost_r$）可由式(12)计算：

$$Cost_r=m\beta \tag{12}$$

特殊地，多服务器系统 MS_j 的租赁费可表示为 $Cost_{r,j}$。

多服务器系统的能耗费可以由单位能耗的费用乘以系统消耗的电能。在本文中，我们重点关注多服务器系统处理器级的能耗。尽管多服务器系统的能耗可能来源于许多方面，但根据文献[22-24]的研究可知，多服务器系统的能耗仍然由处理器主导。设计良好的 CMOS 处理器所消耗的能量可以通过一些简单的模型准确计算，即动态能量消耗模型和静态能量消耗模型。计算动态能耗计算式为 $N_{sw}C_Lf^3$，其中，N_{sw} 是 CMOS 处理器的平均栅极开关因子，C_L 是处理器负载电容，f 是时钟频率。本文假设服务器速度 s 与其处理器的时钟频率 f 成线性比例，即 $s\propto f$。综上，为了简便起见，我们将动态能耗计算公式重新表述为 ξs^3，其中 ξ 可以理解为与处理器相关的常系数。又因为前文中指出，多服务器系统中的云服务请求总是随机的、间歇性的，因此一个多服务器系统的性能不可能总是被云服务请求占满，则此时单位时间内拥有 m 个服务器的多服务器系统的动态能耗可计算为 $m\rho\xi s^3$。多服务器系统的静态能耗是系统空载时所需的待机能耗，通常消耗固定的能耗，用 E^* 来表示，则一个多服务器系统单位时间内消耗的静态能耗为 mE^*。令 γ 表示能源价格，则一个多服务器系统 MS_j 单位时间内消耗的能耗话费是动态能源成本和静态能源成本之和，计算式为：

$$Cost_{ut,j}(m_j,s_j)=m_j\gamma(\rho_j\xi s_j^3+E^*) \tag{13}$$

使用 $Cost_{\text{tot},j}$ 表示多服务器系统 MS_j 单位时间内的总花费，计算式为：

$$\begin{aligned}Cost_{\text{tot},j}(m_j,s_j)&=Cost_{r,j}+Cost_{ut,j}\\&=m_j(\beta+\gamma(\rho_j\xi s_j^3+E^*))\end{aligned} \tag{14}$$

3.5　软错误可靠性模型

如前文所述，当云服务请求在基于 CMOS 技术的服务器上运行时，会遭受瞬时故障（即软错误）的侵袭，而目前有大量的工作探索了面向实时任务处理的软错误可靠性[25-27]的研究。然而，针对多服务器系统中具有随机特性的软实时任务的软错误可靠性研究逐渐受到研究者的关注[26]。与前期工作类似[26]，本文中的云服务请求是具有软实时要求的间歇性云服务，因此我们需要对软实时云服务请求建立软错误可靠性模型来量化分析其是如何影响多服务器配置的。具体建模过程如下所述。

令 λ_{se} 表示服务器的软错误发生率（即在单位时间内服务器遭受软错误的平均数量），其软错误发生率可建模为[25-27]：

$$\lambda_{se}(s)=\lambda_0\mathrm{e}^{d_0\frac{s_{\max}-s}{s_{\max}-s_{\min}}} \tag{15}$$

其中，$s_{\max}$ 和 $s_{\min}$ 分别表示服务器运行的最大及最小速度；λ_0 表示服务器以最大速度运行时的软错误发生率；d_0 是一个常数，用于反映动态电压缩放对软错误发生率的敏感性。

本文作者的前期工作表明[19]，当服务请求在服务器上运行时，服务请求的软错误可靠性可被定义为处理云服务请求不发生软错的概率，即在服务器中任务顺利执行的概率。根据式(15)，服务器的软错误可靠性可计算为：

$$R(m,s,r)=\mathrm{e}^{-\lambda_{se}(s)\cdot\frac{r}{ms}} \tag{16}$$

与文献[28]一样，本文中每个服务请求在服务器上运行时最多可以容忍一个暂时性故障，并且由于软错误而使用重新执行来提高系统的可靠性。需要注意的是，如果服务请求本身及其重新执行都失败，则被视为服务请求处理失败，将不收取用户的费用。此时，重新执行容错机制下的云服务请求的软错误可靠性（R_{re}）可以通过式(17)获得[29]。

$$R_{re}(m,s,r)=1-(1-R(m,s,r))^2 \tag{17}$$

根据式(17)和公式 $f_r(z)$，可推导出处理任意云服务请求的期望软错误可靠性为：

$$SER(m,s)=ms\cdot\left(\frac{2}{ms+\lambda_{se}(s)\cdot\bar{r}}-\frac{1}{ms+2\lambda_{se}(s)\cdot\bar{r}}\right) \tag{18}$$

其具体推导过程可参见文献[28]。

3.6　多服务器系统配置与任务调度问题模型

本节将从云服务提供商的角度出发：CSP 从 CIP 租赁云服务器基础设施资源，为多种类型的应用域构建多服务器系统，向 CC 提供云服务。接下来本文将介绍如何面向异构服务器，将多服务器系统配置与任务调度问题设计成一个统一的云服务利润最大化问题。

云服务提供商为 N 个应用程序域提供服务，每个应用程序域由一个二元组构成 $(\lambda_i,\bar{r}_i)$，其中 λ_i 是单位时间内应用程序域 AD_i 到达云服务请求的平均数量，且集合 $\lambda=\{\lambda_i\mid\lambda_1,\lambda_2,\cdots,\lambda_N\}$，$\bar{r}_i$ 是应用程序域 AD_i 的服务请求所需处理的平均指令数。构建一个多服务器系统 MS 需要根据云服务请求确定选择何种性能的计算单元（即服务器运行速度 s）及计算单元的规模（即服务器数量 m）。每种服务器类型由一个三元组构成 $(sArr_j,Num_j,\beta_j)$，其中 $sArr_j$ 是服务器类型 j 的预定义速度集，即 $sArr_j=\{s_{j,\min},s_{j,2},s_{j,3},\cdots,s_{j,\max}\}$，$Num_j$ 是 j 型服务器的数量，β_j 是服务器类型 j 的租赁单价，由 j 型服务器构成的多服务器系统 MS_j 的配置可由二元组 (m_j,s_j) 表示，且集合 $MS=\{MS_j\mid MS_1,MS_2,\cdots,MS_M\}$。

当云服务请求集 λ 在多服务器系统集 MS 中调度时，理想情况下（即不考虑发生软错误的情况下），云服务提供商处理 AD_i 的期望收益（Rev_i）表示为：

$$Rev_i=\sum_{i=1}^{N}\sum_{j=1}^{M}(x_i^j\cdot\bar{C}_i(\lambda_i,m_j,s_j)\cdot\lambda_i) \tag{19}$$

其中，$x_i^j=0$ 或 1，当其取 1 时，表示将应用域 AD_i 调度至多服务器系统 MS_j，反之则亦然，且 $\sum_{i=1}^{N}\sum_{j=1}^{M}x_i^j=N$，$\sum_{j=1}^{M}x_i^j=1(1\leqslant i\leqslant N)$。

特别地，令 $\sum_{i=1}^{N}\sum_{j=1}^{M}x_i^j\cdot\lambda_i=\lambda_{i,j}$。

当 $x_i^j=1$ 时，式(19)可继续展开为：

$$\lambda_i\cdot a\bar{r}_i\left(1-\frac{P_b^j}{(m_js_j-\lambda_{i,j}\bar{r}_i)\left(\frac{c}{s_0}-\frac{1}{m_js_j}\right)+1}\cdot\right.$$

$$\frac{1}{(m_j s_j-\lambda_{i,j}\bar{r}_i)\left(\frac{a}{d}+\frac{c}{s_0}-\frac{1}{m_j s_j}\right)+1}\Bigg) \tag{20}$$

当处理应用程序域 AD_i 时，在考虑云服务请求可能会受到软错误影响的情况下，处理 AD_i 的期望收益（$Rev_{ser,i}$）可由式(21)计算：

$$Rev_{ser,i}=\sum_{i=1}^{N}\sum_{j=1}^{M}(x_i^j\cdot\bar{C}_i(\lambda_i,m_j,s_j)\cdot\lambda_i\cdot SER_j) \tag{21}$$

则根据式(19)和式(21)，云服务请求调度与多服务器系统配置问题可统一成云服务利润最大化问题，可表示为：

$$Max:Profit=\sum_{i=1}^{N}\sum_{j=1}^{M}(x_i^j\cdot\bar{C}_i\cdot\lambda_i\cdot SER_j-Cost_{tot,j}) \tag{22}$$

$$\text{s.t.}\sum_{i=1}^{N}\sum_{j=1}^{M}x_i^j=N \tag{23}$$

$$x_i^j=0\text{ 或 } \tag{24}$$

$$\sum_{j=1}^{M}x_i^j=1(1\leqslant i\leqslant N) \tag{25}$$

$$\rho_j<1 \tag{26}$$

$$\sum_{i=1}^{N}x_i^j\cdot m_j<Num_j(1\leqslant j\leqslant M) \tag{27}$$

其中，式(23)－式(25)表示任意应用域 AD_i 只能被调度到其中一个 MS_j 中，式(26)表示多服务器系统的服务率必须小于1，否则会引起系统阻塞，式(27)表示服务器个数的约束。

4 服务请求调度与多服务器系统配置方法

本章将为地理分布式的三层云服务结构提出一个新颖的云服务请求调度与多服务器系统配置方法，该方法主体基于灰狼算法来设计云服务请求调度策略与多服务器系统配置策略。我们的方法可以最大化地利用异构的云服务计算资源，具体包含一个基于深度搜索的随机初始化算法以及一个回溯算法。基于本文所提的服务请求调度与多服务器配置方法可以最大化云服务提供商的利润。详细介绍如下。

4.1 基于灰狼算法的云服务利润最大化策略

式(22)－式(25)构成一个含约束的非线性优化问题，也是一个复杂的组合优化问题，该问题无法通过直接计算来求得其封闭解[2]。灰狼算法[28]作为一种新颖的群体智能算法，调节参数少，迭代过程简单，其将全局搜索与局部启发式搜索相结合，可使算法避免陷入局部最优。本文基于灰狼算法设计了一个云服务利润最大化机制，可以协同优化云服务请求调度和多服务器系统配置，该机制不仅可以最大化云服务利润，还可以充分利用异构的云计算资源，避免资源浪费。该机制由算法1所示。

算法1 基于灰狼算法的云服务利润最大化策略

输入参数：

①云服务应用域：$(\lambda_i,\bar{r}_i)(i=1,2,\cdots,N)$，即$\boldsymbol{\lambda}_{arr}=\{\lambda_i\mid\lambda_1,\lambda_2,\cdots,\lambda_N\}$，$\mathbf{r}_{arr}=\{\bar{r}_i\mid\bar{r}_1,\bar{r}_2,\cdots,\bar{r}_N\}$；

②服务器：$(sArr_j,Num_j,\beta_j)(j=1,2,3,\cdots,M)$，即$\mathbf{s}_{arr}=\{sArr_j\mid sArr_1,sArr_2,\cdots,sArr_M\}$，$\mathbf{m}_{arr}=\{Num_j\mid Num_1,Num_2,\cdots,Num_M\}$，$\boldsymbol{\beta}_{arr}=\{\beta_j\mid\beta_1,\beta_2,\cdots,\beta_M\}$；

输出：

①云服务请求调度策略：$x_i^j(1\leqslant i\leqslant N,1\leqslant j\leqslant M)$；

②MS配置策略：$(m_{i,j},s_{i,j})(1\leqslant i\leqslant N,1\leqslant j\leqslant M)$，最大云服务利润：Profit

1. 初始化一个狼群中灰狼个数：$wof_{num}=30$；
2. 设置灰狼算法最大迭代次数 I=30；//灰狼算法寻找最优解的最大迭代次数
3. 定义灰狼个体编码（即可行解）：wof [1,N*2]；//将灰狼个体编码为一个一维数组，第1～N位用来表示 $\lambda_i(i=1,2,\cdots,N)$对应的服务器类型，第N+1～N*2位用来表示与各 λ_i对应服务器类型的个数
4. 初始化3个狼群：$wof_{group}[wof_{num}*3,N*2]$；//此步骤是为了避免灰狼算法陷入局部最优，初始化3个狼群，并调用RABDS算法来初始化，将初始化结果按照 Profit 的大小从大到小进行排序
5. 初始化 α 狼，β 狼，σ 狼 //调用RABDS算法随机初始化3个可行解
6. For i=1:I
7. For j=wof_{num}+1:wof_{num}*2//使 wof_{group} 中第 wof_{num}+1～wof_{num}*2 头灰狼寻找猎物（即潜在最优解）
8. For k=1:N*2
9. init r_1,r_2;//初始化搜索半径
10. a=2−t*(2/(N*2));//收敛系数
11. $A_1=2*a*r_1-a$;//计算系统系数
12. $C_1=2*r_2$;//计算协同系数
13. $D_\alpha=\|C_1*\overrightarrow{wof_\alpha}-\overrightarrow{wof_{group}(j)}\|$;//计算 α 与猎物之间的距离
14. $X_1=\overrightarrow{wof_\alpha}-A_1*D_\alpha$;//更新 α 狼
15. init r_1,r_2;
16. $A_1=2*a*r_1-a$;
17. $C_1=2*r_2$;
18. $D_\beta=\|C_1*\overrightarrow{wof_\beta}-\overrightarrow{wof_{group}(j)}\|$;//计算 β 与猎物之间的距离
19. $X_2=\overrightarrow{wof_\beta}-A_1*D_\beta$;//更新 β 狼
20. init r_1,r_2;
21. $A_1=2*a*r_1-a$;
22. $C_1=2*r_2$;
23. $D_\sigma=\|C_1*\overrightarrow{wof_\sigma}-\overrightarrow{wof_{group}(j)}\|$;//计算 σ 与猎物之间的距离
24. $X_3=\overrightarrow{wof_\sigma}-A_1*D_\sigma$;//更新 σ 狼
25. $wof_{group}(j,k)=(X_1+X_2+X_3)/3$;//更新所有的狼的位置
26. For j=wof_{num}*2+1:wof_{num}*3 //添加扰动因素避免灰狼算法陷入局部最优
27. $wof_{group}(j)$=RABDS();//利用随机算法对 wof_{num}*2+1:wof_{num}*3 头狼进行赋值
28. 利用式(22)计算所有狼的 Profit，并将其从大到小排序；
29. 输出：第一个 $wof_{group}[1,N*2]$数组及所获云服务利润 Profit;//将寻找到的最优解输出，数组中包含云服务请求调度策略和多服务器配置策略

算法1使用云服务应用域：$(\lambda_i,\bar{r}_i)(i=1,2,\cdots,N)$及服务器集：$(sArr_j,Num_j,\beta_j)(j=1,2,3,\cdots,M)$作为输入。然后，算法1首先初始化狼群规模为30头狼，即$wof_{num}=30$，设置灰狼算法的最大迭代次数 $I=30$，将灰狼个体定义为一个1行 $N*2$ 列的一维数组，其中1～N 位用来存放 $\lambda_i(i=1,2,\cdots,N)$对应的服务器类型，第 $N+1$～$N*2$ 位用来存放与各 λ_i对应服务器类型的个数（第1－3行）。为了避免灰狼算法陷入局部最优，调用RABDS算法初始化3个狼群，将初始化结果按照 $Profit$ 的大小从大到小排序，选取前30头狼作为狩猎狼群，并初始化3只头狼，即 α 狼，β 狼，σ 狼（第4－5行）。算法利用狩猎狼群来寻来猎物，即最优解（第6－25行）。α 狼寻找猎物的过程，可分为：初始化搜索半径（第9行），计算算法关键系数（第10－12行），其中，a 为收敛系数，用来控制算法进行线性收敛，A_1 和 C_1 为算法的协同系数，用来计算 α 狼与猎物之间的距离并更新其位置。β 狼与 σ 狼寻找猎物的过程与 α 狼类似（第15－24行）。3只头狼寻找到猎物后，更新狼群的位置（第25行）。添加扰动因素，避免算

法陷入局部最优（第 26－27 行）。将所有灰狼寻找到的猎物（即最优解）按 Profit 从大到小排序并输出第一只狼的结果（第 28－29 行）。

由于灰狼算法在寻找最优解时需要基于初始化的可行解，可行解越优越容易找到最优解，本文基于云服务调度与多服务器配置的利润最大化问题是一个复杂的组合优化，无法通过随机赋值的形式找到较优的可行解，因此本文设计了一个基于深度搜索的初始化算法来确保初始化的解满足本文优化问题的约束。算法 1 中调用的 RABDS 算法进行灰狼个体初始化，细节将由算法 2 给出，算法 2 的输入即为算法 1 的输入，该算法首先初始化灰狼个体为空（即数组 ***res***）（第 1 行），初始化应用程序域与服务器类型数量的映射关系数组 $\boldsymbol{m}_{\text{lowbound}}$ 为空（第 2 行）。将标志位的默认值设为 1（第 3 行），这是调用回溯算法 backtrackInit() 的默认输入，回溯算法具体将由算法 3 给出。接着，算法 2 将对数组 $\boldsymbol{m}_{\text{lowbound}}$ 进行赋值（第 4～6 行），并调用溯算法 backtrackInit() 给数组 ***res*** 的第 1～N+1 位赋值，其中第 1～N 位为应用程序域与服务器类型的映射关系，第 N+1 位为标示位，用来表示是否存在可行解（第 7 行），如果第 N+1 位为 0 则说明不存在可行解（第 8－9 行）。有了可行的映射关系后，算法首先给各应用程序域配置最少个数的服务器（第 10－12 行）；接着随机选择一个应用程序域（第 13 行），使用剩余服务器对其进行再次配置（第14－18 行）；最后，返回满足约束（即式（23）－式（27））的灰狼个体 ***res***（即一个可行解）（第 19 行）。

算法 2　基于深度搜索的随机初始化算法（RABDS）

输入参数：

①云服务应用域：$(\lambda_i, \bar{r}_i)(i=1,2,\cdots,N)$，即 $\lambda_{arr}=\{\lambda_i \mid \lambda_1,\lambda_2,\cdots,\lambda_N\}$，$r_{arr}=\{\bar{r}_i \mid \bar{r_1},\bar{r_2},\cdots,\bar{r_N}\}$

②服务器：$(sArr_j, Num_j, \beta_j)(j=1,2,3,\cdots,M)$，即 $s_{arr}=\{sArr_j \mid sArr_1, sArr_2, \cdots, sArr_M\}$，$m_{arr}=\{Num_j \mid Num_1, Num_2, \cdots, Num_M\}$，$\beta_{arr}=\{\beta_j \mid \beta_1,\beta_2,\cdots,\beta_M\}$

输出：

①灰狼个体 **res**

1. 初始化数组 res[1，N＊2]＝NULL；//灰狼
2. 初始化空数组 $m_{lowbound}[N,M]$＝NULL；//用来存储当应用程序域 i 调度至服务器类型 j 时，所需的最少服务器数量
3. Setindex＝1；//给回溯算法设置标志位
4. For i＝1∶N
5. 　For j＝1∶M
6. 　　$m_{lowbound}(i,j)=\dfrac{\lambda_i \cdot \bar{r}_i}{s_{max,i}}$；//使用式（1）计算当选择服务器类型 j 处理应用域 i 时，至少需要多少个服务器
7. **res**＝backtrackInit(N，index，λ_{arr}，$\mathbf{r}_{arr}$，$\mathbf{m}_{arr}$，$\mathbf{s}_{arr}$，**res**，$\mathbf{m}_{lowbound}$)；//调用回溯法
8. if λ(N＋1) ＝＝ 0
9. 　return “没有可行方案”；
10. for i＝1∶N //配置最少个数的服务器数量
11. 　m_{index}＝ λ(i)；//获取 λ_i 的 server 类型
12. 　$m_{arr}(m_{index})=m_{arr}(m_{index})-m_{lowbound}(i,m_{index})$；//扣除当前 server 需要的最少个数，以满足处理应用域的最低需求
13. $rand_{index}$＝random(1，N)；//从 1～N 随机寻找一个起始 λ 的下标
14. for i＝1∶N //二次配置
15. 　m_{index}＝ res($rand_{index}$)；//随机选在一个应用域开始二次分配
16. 　res(N＋$rand_{index}$) ＝res(N＋$rand_{index}$)＋ random(0，m_{arr}(m_{index}))；
17. 　$m_{arr}(m_{index})=m_{arr}(m_{index})-$res(N＋$rand_{index}$)；
18. 　$rand_{index}$＝ mod($rand_{index}$，N)＋1；
19. Return **res**；

算法 2 使用回溯算法（详见算法 3）来寻找应用程序域与多服务器系统可行的调度关系，算法 3 的具体描述如下。算法 3 的输入除了算法 2 的输入外，还有标示位 *index*（用来标示当前需要调度的应用程序域），***res*** 灰狼个体，其中，前 1～N 位表示需要初始化的应用域和服务器类型对应关系，第 N＋1 位表示算法的标识位（该标识用来给算法 1 判断配置多服务器数量的起始位），其余位空置，以及应用程序域与服务器类型数量的映射关系数组 $\boldsymbol{m}_{\text{lowbound}}$。该算法首先为应用程序域 λ_{index} 随机选择服务器类型 $index_m$（第 1 行），并寻找应用域的可行调度策略（第 2－16 行）。在寻找可行的调度策略过程中，如果当前类型的服务器数量余值超过 λ_{index} 所需的最小数量，则将 λ_{index} 调度至该类型的多服务器，并更新剩余服务器数量（第 3－5 行），直至为最后一个 λ 寻找到可行的调度策略，将表识位置 1 后返回 ***res***（第 6－8 行），否则继续为下一个 λ 寻找可行的调度策略直至结束（第 9－12 行）。若无法为所有应用程序域制定可行的调度策略，则说明基于 $index_m$ 类型服务器的初始调度方案不存在可行解，需进行回溯（第 13－15 行），并为 λ_{index} 重新寻找初始调度方案（第 16 行），直至为初始 λ_{index} 遍历完所有类型的服务器都不能形成可行的调度策略，则将标识位置 0 返回（第 17－18 行）。

算法 3　回溯算法（backtrackInit）

输入参数：

①云服务应用域：$(\lambda_i, \bar{r}_i)(i=1,2,\cdots,N)$，即 $\boldsymbol{\lambda}_{arr}=\{\lambda_i \mid \lambda_1,\lambda_2,\cdots,\lambda_N\}$，$\mathbf{r}_{arr}=\{\bar{r}_i \mid \bar{r_1},\bar{r_2},\cdots,\bar{r_N}\}$；

②服务器：$(sArr_j, Num_j, \beta_j)(j=1,2,3,\cdots,M)$，即 $\mathbf{s}_{arr}=\{sArr_j \mid sArr_1, sArr_2, \cdots, sArr_M\}$，$\mathbf{m}_{arr}=\{Num_j \mid Num_1, Num_2, \cdots, Num_M\}$，$\boldsymbol{\beta}_{arr}=\{\beta_j \mid \beta_1,\beta_2,\cdots,\beta_M\}$；

③标志位：index；灰狼个体 **res**；数组 $\mathbf{m}_{lowbound}$

输出：

①灰狼个体 **res**

1. $index_m$＝ random(1，length($\mathbf{m}_{arr}$))；//为当前应用域 λ_{index} 随机选取服务器类型
2. for i＝1∶length($\mathbf{m}_{arr}$)
3. 　if $m_{lowbound}(index,index_m) \leqslant m_{arr}(index_m)$//（随着迭代会扣除）剩余的机器数
4. 　　res(index) ＝$index_m$；
5. 　　$m_{arr}(index_m)=m_{arr}(index_m)-m_{lowbound}(i,index_m)$；//减去使用掉的最少的机器数量
6. 　　if index ＝＝ length($\boldsymbol{\lambda}_{arr}$) //判断一下是否分配到最后一个 λ
7. 　　res(N＋1)＝1；
8. 　　return **res**；
9. 　　else
10. 　　**res** ＝ backtrackInit(index ＋ 1，$\boldsymbol{\lambda}_{arr}$，$\mathbf{r}_{arr}$，$\mathbf{m}_{arr}$，$s_{arr}$，**res**，$\mathbf{m}_{lowbound}$) //调度没有结束继续调度下一个应用程序域
11. 　　if res(N＋1) ＝＝ 1
12. 　　return **res**；
13. 　　else
14. 　　res(index)＝0；//基于 $index_m$ 类型服务器的初始调度方案，无法为所有应用程序域制定可行的调度策略，进行回溯
15. 　　$m_{arr}(index_m)=m_{arr}(index_m)+m_{lowbound}(index,index_m)$；//加上当前需要的最少机器数
16. 　　$index_m$＝mod($index_m$，length($\mathbf{m}_{arr}$)) ＋ 1；//寻找下一个 $index_m$

17. res(N+1)=0;
18. return res;

5 实验与评估

本文考虑了 CSP 向 CIP 租用基础设施资源配置多服务器系统来处理 CC 提交的服务请求的情况，并对多核处理器级的多服务器系统开展配置优化工作。假设一个多服务器系统即为一个多核服务器处理器，并利用 AMD EPYC 7742 处理器[30]、Intel Platinum 8376 处理器[31]等高级处理器来模拟我们的多服务器系统。表 1 列出了用于估计多服务器单位能耗成本、基础设施资源使用成本和软错误相关的参数。

表 1 本文主要符号的定义

Table 1 Definition of main notations in this paper

Notation	Definition	Value
a	执行云服务请求单位指令数量的费用	20 cents
d	与服务质量相关的惩罚因子	1
c	一个常量	3
s_0	用户处理云服务请求的期望数量	1.0
γ	单位能源的价格	0.1 cents/Watt·s
E^*	服务器的静态能耗	2(Watts/s)
ξ	处理器相关的一个系数	9.4192
λ_0	服务器以最大速度运行时的软错误发生率	10^{-4}
d_0	一个常数，用来反映动态电压缩放对软错误的敏感性	2

本章中进行了 3 组仿真实验，从不同的角度验证了所提出方案。在第一组实验中，将对比是否考虑软错误时多服务器系统的配置及云服务利润，本实验旨在展示不考虑软错误可靠性的情况下，多服务器系统服务云服务请求造成的性能损失。在第二组实验中，与最新的多服务器配置方法(FCMS)[2]进行了多服务器系统的配置及云服务利润的比较。本实验旨在展示在不同资源利用策略下多服务器系统配置及云服务利润的情况。在第三组实验中，将所提出的利润最大化方案与一种基准测试方法(MCRDF)[29]进行了比较。该实验旨在证明所提出的方案在提高云服务利润方面的有效性。所有实验都在同一台机架服务器上实现，该服务器安装了 Linux 版本的 Matlab x64(版本 9.3.0.713579，包含 Simulink，Global Optimization Toolbox 等)，并配备 64GB DDR4 内存和 2.1GHz Intel Xeon Silver 8 核处理器。

5.1 多服务器系统配置及利润与无 SER 的比较

在本节实验中，我们旨在探索当不考虑/考虑软错误发生的情况下，本文方法所得到的多服务器系统配置及相应云服务利润的比较，以验证本文基于软错误可靠性模型的有效性。在本实验中，我们将云服务请求应用域设置为：$\{(\lambda_i,\bar{r}_i)|(20,1.0),(20,1.1),(40,1.2),(30,1.2),(60,1.3),(80,1.5)\}$，其他参数如 a,d,c,s_0 等系统参数如表 1 所列，服务器类型参数如表 2 所列。

表 2 不同服务器类型集

Table 2 Sets of different server types

Server Type j	Server Speed Set	Maximum Number	Rental Price β
1	[4.5,4.7,4.9,5.1,5.4]	18	1.8 cents/s
2	[4.3,4.5,4.7,5.0,5.2]	24	1.8 cents/s
3	[3.0,3.2,3.6,3.9,4.1]	30	1.6 cents/s
4	[3.0,3.3,3.7,4.0,4.1]	36	1.6 cents/s
5	[2.4,2.6,2.8,3.1,3.4]	40	1.4 cents/s
6	[2.3,2.5,2.7,3.0,3.3]	56	1.4 cents/s
7	[2.0,2.2,2.5,3.0,3.2]	70	1.2 cents/s
8	[1.8,2.0,2.3,2.5,2.7]	80	1.0 cents/s

表 3 中的数据比较了在处理云服务请求时，各应用程序域与多服务器系统的调度关系及其配置、云服务最大利润。在考虑软错误可靠性的情况下，各应用域分别选择服务器类型为：{6,7,8,7,5,5}，对应的最大利润为 4715.6502；当不考虑软错误可靠性时，各应用域分别选择服务器类型为：{8,7,4,7,1,5}，对应的最大利润为 4418.7643，此时会损失约 6.72%的利润。考虑到本文利润的计量单位为 cent(美分)，属于较小的单位，当基数较大时损失的利润也会增加，因此该部分利润的损失是不可忽视的。造成利润损失的主要原因在于后者没有考虑 SER 对云服务利润的影响，从而影响了应用程序域的调度与多服务器系统配置策略，如图 3 所示，当不考虑 SER 对云服务系统的影响时，得到的多服务器系统配置的平均软错误发生的概率将降低约 17.89%。

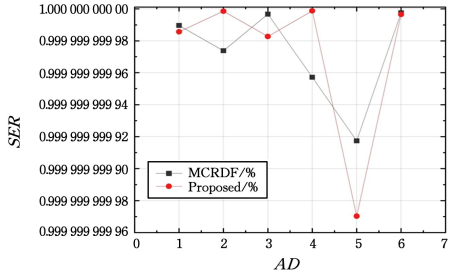

图 3 在考虑/不考虑软错误下所提方法的 SER

Fig. 3 SER of the proposed method with/without soft error

表 3 在考虑/不考虑 SER 情况下的利润比较

Table 3 Profit comparison of the proposed method with/without SER

AD_i	Proposed Method with SER					Proposed Method without SER				
	server type j	m	s	$profit$	$Profit_{tot}$	server type j	m	s	$profit$	$Profit_{tot}$
AD_1	6	56	3.3	1712.2540	4715.6502	8	71	2.7	1948.3104	4418.7643
AD_2	7	36	3.2	1143.3590		7	35	3.2	1051.7161	
AD_3	8	48	2.7	755.9126		4	20	4.1	517.0664	
AD_4	7	20	3.2	556.0326		7	19	3.2	556.1084	
AD_5	5	12	3.4	301.4012		1	6	5.4	8.2429	
AD_6	5	18	3.4	246.6908		5	10	3.4	275.3271	

5.2 多服务器系统配置及利润与 FCMS 方法的比较

在本节实验中，我们旨在将本文提出的多服务器系统配置与云服务请求调度方法和该领域最新的方法(FCMS)在优化云服务利润方面进行比较，以验证本文方法的有效性。FCMS 是一个资金敏感型多应用域云服务利润最大化策略，该策略综合云服务请求数量、服务水平协议等因素来配置多服务器系统以最大化云服务利润。在本实验中，我们将云服务请求应用域设置为：$\{(\lambda_i,\bar{r}_i)|(20,1.0),(20,1.1),(40,1.2),(30,1.2),(60,1.3),(80,1.5)\}$，系统参数如 a,d,c,s_0 等如表 1 所列，服务器类型参数如表 2 所列。

表 4 中的数据比较了在处理云服务请求时,各应用程序域与多服务器系统的调度关系及其配置、云服务最大利润,本文方法的各应用域分别选择的服务器类型为:{6,7,8,7,5,5},其相应的最大云服务利润为 4715.6502,而 FCMS 的各应用域分别选择的服务器类型为:{6,7,5,8,1,4},其相应的最大云服务利润为 4 206.989 1。本文方法可提升利润约 12.09%。与 FCMS 不同的是,本文方法在进行应用程序域调度和多服务器系统配置时考虑了软错误对利润的影响。如图 4 所示,本文方法比 FCMS 的软错误平均发生率低约 19.57%。此外,本文方法还充分考虑了异构服务器资源的利用,而在 FCMS 中则是将所有的同类服务器构建成一个多服务器系统,这会造成一定程度的计算资源浪费。

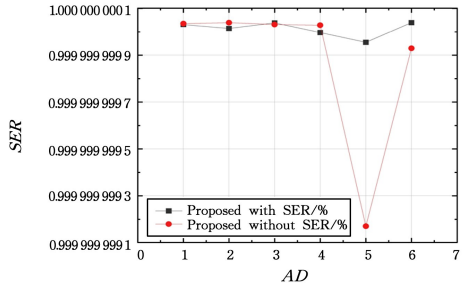

图 4　本文方法与 FCMS 的 SER 比较

Fig. 4　SER comparison between the proposed method and FCMS

表 4　本文方法与 FCMS 的利润比较

Table 4　Profit comparison between the proposed method and FCMS

AD_i	the Proposed Method					FCMS				
	server type j	m	s	$profit$	$Profit_{tot}$	server type j	m	s	$profit$	$Profit_{tot}$
AD_1	6	56	3.3	1712.2540		6	50	3.3	1612.6594	
AD_2	7	36	3.2	1143.3590		7	63	3.2	1177.9210	
AD_3	8	48	2.7	755.9126	4715.6502	5	39	3.4	637.1780	4206.9891
AD_4	7	20	3.2	556.0326		8	40	2.7	526.1345	
AD_5	5	12	3.4	301.4012		1	6	5.4	8.2429	
AD_6	5	18	3.4	246.6908		4	12	4.1	208.8534	

5.3　多服务器系统配置及利润与 MCRDF 方法的比较

在本节实验中,我们旨在将本文方法得到的多服务器系统配置与云服务请求调度方法和该领域最新的方法(MCRDF)在多服务器系统配置与云服务利润方面进行比较,以验证本文方法的有效性。MCRDF 基于遗传算法对多服务器系统配置及云服务请求调度进行了研究,通过配置多服务器系统最大化云服务利润。在本实验中,我们将云服务请求应用域设置为:$\{(\lambda_i,\bar{r}_i)|(20,1.0),(20,1.1),(40,1.2),(30,1.2),(60,1.3),(80,1.5)\}$,系统参数如 a,d,c,s_0 等如表 1 所列,服务器类型参数如表 2 所列。

表 5 中的数据比较了在处理云服务请求时,各应用程序域与多服务器系统的调度关系及其配置、云服务最大利润。本文方法的各应用域分别选择的服务器类型为:{6,7,8,7,5,5},其相应的最大云服务利润为 4715.6502,而 MCRDF 的各应用域分别选择的服务器类型为:{8,7,6,4,5,6},其相应的最大云服务利润为 4 638.135 0。本文方法可提升利润约 1.67%。MCRDF 是利用遗传算法来配置多服务器系统和进行云服务调度,该方法给出的多服务器系统配置策略能得到与本文方法接近的软错误可靠性,如图 5 所示,但本文方法给出的调度策略要优于 MCRDF,可以获得小幅的利润提升。

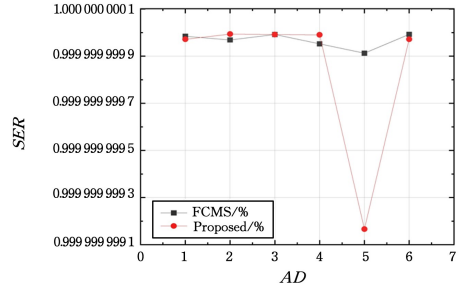

图 5　本文方法与 MCRDF 的 SER 比较

Fig. 5　SER comparison between the proposed method and MCRDF

表 5　本文方法与 MCRDF 的利润比较

Table 5　Profit comparison between the proposed method and MCRDF

AD_i	the Proposed Method					MCRDF				
	server type j	m	s	$profit$	$Profit_{tot}$	server type j	m	s	$profit$	$Profit_{tot}$
AD_1	6	56	3.3	1712.2540		8	65	2.7	1869.7372	
AD_2	7	36	3.2	1143.3590		7	61	3.2	1180.7210	
AD_3	8	48	2.7	755.9126	4715.6502	6	27	3.3	677.6179	4638.1350
AD_4	7	20	3.2	556.0326		4	36	4.1	350.0179	
AD_5	5	12	3.4	301.4012		5	11	3.4	302.8962	
AD_6	5	18	3.4	246.6908		6	27	3.3	257.1449	

结束语　本文在云服务三层架构中,为云服务提供商设计了一个资源配置与云服务请求调度方法,该方法充分考虑了计算资源的异构性与云服务请求的多样性,可充分利用性能异构的服务器资源。并在现有云服务的服务水平协议基础上,考虑了云服务请求在执行时遭受软错误的情况,建立了软错误容错机制,并在配置多服务器系统时充分考虑了软错误对云服务利润的影响。最终将面向软实时云服务请求的计算资源配置与请求调度方法形式化成一个利润最大化问题,并为该问题设计了一个基于灰狼算法的多服务器系统配置方法,大量的实验结果表明,我们的方法与基准方法相比可以平均提升 6.83%的利润。

参 考 文 献

[1]　MA X J,RAO G B,XU H H. Research on Task Scheduling in Cloud Computing[J]. Computer Science,2019,46(3):1-8.

[2]　ZHOU M S,DONG X S,CHEN H,et al. Improving Cloud Plat-

form Based on the Runtime Resource Capacity Evaluation[J]. Journal of Computer Research and Development, 2017, 54(11): 2516-2533.
[3] CONG P, LI L, ZHOU J, et al. Developing user perceived value based pricing models for cloud markets[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(12): 2742-2756.
[4] WANG T, ZHOU J, ZHANG G, et al. Customer perceived value and risk aware multiserver configuration for profit maximization[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 13(5): 1074-1088.
[5] SUN D W, CHANG G R, CHEN D, et al. Profiling, Quantifying, Modeling and Evaluating Green Service Level Objectives in Cloud Computing Environments[J]. Chinese Journal of Computers, 2013, 36(7): 1509-1525.
[6] LUČANIN D, PIETRI I, HOLMBACKA S, et al. Performance-based pricing in multi-core geo-distributed cloud computing[J]. IEEE Transactions on Cloud Computing, 2020, 8(4): 1079-1092.
[7] CAO J, HUANG K, LI K, et al. Optimal multiserver configuration for profit maximization in cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1087-1096.
[8] Service Level Agreement [OL]. [2021-12-27] https://en.wikipedia.org/wiki/Service level agreement.
[9] MEI J, LI K, LI K. Customer-Satisfaction-Aware Optimal Multiserver Configuration for Profit Maximization in Cloud Computing[J]. IEEE Transactions on Sustainable Computing, 2017, 2(1): 17-29.
[10] GOUDARZI H, PEDRAM M. Maximizing profit in cloud computing system via resource allocation[C]// Proceedings of IEEE International Conference on Distributed Computing Systems. Minneapolis, MN, 2011: 1-6.
[11] MEI J, LI K, OUYANG A, et al. A profit maximization scheme with guaranteed quality of service in cloud computing[J]. IEEE Transactions on Computers, 2015, 64(11): 3064-3078.
[12] KANG Z, YANG B. A study of optimal multi-server system configuration with variate deadlines and rental prices in cloud computing[C]// Proceedings of Springer International Conference on Human Centered Computing. Cham: Springer, 2017: 215-231.
[13] XU H, LI B. Dynamic cloud pricing for revenue maximization [J]. IEEE Transactions on Cloud Computing, 2013, 1(2): 158-171.
[14] ALI H, SAROIT A, KOTB M. Grouped tasks scheduling algorithm based on QoS in cloud computing network[J]. Egyptian Informatics Journal, 2017, 18(1): 11-19.
[15] CHEN W, XIE G, LI R, et al. Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems[J]. Future Generation Computer Systems, 2017, 74: 1-11.
[16] TIAN J, HU W, WANG Y, et al. A novel PSO based task scheduling algorithm for multi-core systems[C]// Proceedings of International Conference on Smart Computing and Communication. Cham: Springer, 2016: 62-71.
[17] DENG Y, CHENG H. A heterogeneous multiprocessor task scheduling algorithm based on SFLA[C]// Proceedings of World Automation Congress. Rio Grande, PR, 2016: 1-5.
[18] iCloud [OL]. [2023-06-06]. https://support.apple.com/zh-cn/HT208351.
[19] WANG T, ZHOU J, LI L, et al. Deadline and Reliability Aware Multiserver Configuration Optimization for Maximizing Profit [C]// IEEE Transactions on Parallel and Distributed Systems. 2022: 3772-3786.
[20] WANG T, ZHANG M, SHEN W, et al. A multiserver configuration and request distribution framework for profit maximization in a three-tier cloud service architecture[J]. Journal of Circuits, Systems, and Computers, 2022, 31(12): 2250221.
[21] KOBAYASHI H, KONHEIM A. Queueing Models for Computer Communications System Analysis[J]. IEEE Transactions on Communications, 1977, 25(1): 2-29.
[22] SONG J, LI T T, YAN Z X, et al. Energy-Efficiency Model and Measuring Approach for Cloud Computing[J]. Journal of Software, 2012, 23(2): 200-214.
[23] HU Y, LIU C, LI K, et al. Slack allocation algorithm for energy minimization in cluster systems[J]. Future Generation Computer System, 2017, 74: 119-131.
[24] WU T, GU H, ZHOU J, et al. Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud[J]. Journal of Systems and Architecture, 2018, 84: 12-27.
[25] LI J, LIN B, CHEN X. Reliability Constraint-oriented Workflow Scheduling Strategy in Cloud Environment [J]. Computer Science, 2023, 50(10): 291-298.
[26] WU Y H, HUANG G, ZHANG Y, et al. A Model-Based Fault Tolerance Mechanism Development Approach for Cloud Computing[J]. Journal of Computer Research and Development, 2016, 53(1): 138-154.
[27] ZHOU J, LI L, VAJDI A, et al. Temperature-Constrained Reliability Optimization of Industrial Cyber-Physical Systems Using Machine Learning and Feedback Control[C]// IEEE Transactions on Automation Science and Engineering. 2023: 20-31.
[28] MIRJALILI S, MIRJALILI S M, LEWIS A. Grey Wolf Optimizer [J]. Advances in Engineering Software, 2014, 69(3): 46-61.
[29] HAQUE M A, AYDIN H, ZHU D. On reliability management of energy-aware real-time systems through task replication[J]. IEEE Transacations on Parallel Distributed Systems, 2017, 28(3): 813-825.
[30] AMD EPYC7742, 2022[OL]. https://www.amd.com/zhhans/products/cpu/amd-epyc-7742.
[31] Intel Platinum 8376H, 2022[OL]. https://www.intel.cn/content/www/cn/zh/products/sku/204096/intel-xeon-platinum-8376h-processor-38-5m-cache-2-60-ghz/specifications.html.

WANG Tian, born in 1990, Ph.D, lecturer, is a member of CCF (No. A6921M). His main research interests include cloud computing, edge computing, and hyperphysical system.

WANG Zheng, born in 1972, master, professor. His main research interests include computer application technology, big data and e-commerce, and computer systems.