王磊磊,邓晓衡,桂劲松,等. 边缘计算中基于任务分解的任务分配算法[J]. 湖南科技大学学报(自然科学版), 2022, 37(1):76-84. doi:10.13582/j.cnki.1672-9102.2022.01.011

WANG L L, DENG X H, GUI J S, et al. Task Scheduling Algorithm Based on Task Decomposition in Edge Computing [J]. Journal of Hunan University of Science and Technology (Natural Science Edition), 2022, 37(1):76-84. doi:10.13582/j.cnki.1672-9102.2022.01.011

边缘计算中基于任务分解的任务分配算法

王磊磊[1],邓晓衡[1,2*],桂劲松[1],刘斌[3],付琨[3]

(1.中南大学 计算机学院,湖南 长沙 410075;2.中南大学 深圳研究院,广东 深圳 518071;
3.中国科学院 航天信息研究所,北京 100000)

摘 要:大规模任务使得任务服务质量遭受到巨大挑战.边缘计算环境能够为大规模任务处理提供很好的执行模式.针对这一问题,提出了一种基于任务分解的任务调度算法(Task Decomposition based task Allocation Friendly algorithm, TDAF).该算法主要包含 2 个模块:任务分解模块和任务调度模块.在任务分解模块中,设计了任务关联矩阵以及定义了和任务关联性相关的定义介绍,进而通过任务间的相似度和信息输入输出间的关联性得到任务关联性的计算.以最小化任务关联性设计任务分解目标函数,采用遗传算法对任务进行优化分解.在任务调度模块,通过计算分解后的子任务的优先级确定每一个子任务资源分配的方案.仿真结果表明:TDAF 算法的任务执行完成时间和吞吐率性能更优.

关键词:边缘计算;任务分解;任务分配;任务关联性;遗传算法;优先级

中图分类号:TP305 **文献标志码**:A **文章编号**:1672-9102(2022)01-0076-09

Task Scheduling Algorithm Based on Task Decomposition in Edge Computing

WANG Leilei[1], DENG Xiaoheng[1,2], GUI Jingsong[1], LIU Bin[3], FU Kun[3]

(1. School of Computer Science and Engineering, Central South University, Changsha 410075, China;
2. Research Institute of Shenzhen, Central South University, Shenzhen 518071, China;
3. Institute of Aerospace Information, Chinese Academy of Sciences, Beijing 100000, China)

Abstract: Large-scale task makes task service quality suffer from a huge challenge. Edge computing environments can provide a good mode of execution for large-scale task processing. To address this problem, we proposes a novel Task Decomposition based task Allocation Friendly algorithm (TDAF). The algorithm mainly consists of two modules: task decomposition module and task scheduling module. In the task decomposition module, we design the task association matrix and define the definition introduction related to the task association, and then obtain the calculation of task association through the similarity between tasks and the association between information input and output. In order to minimize task association, the objective function of task decomposition is designed, and genetic algorithm is used to optimize task decomposition. In the task scheduling module, we determine the resource allocation scheme for each subtask by calculating the priority of the decomposed subtasks. The simulation results show that the TDAF algorithm has better completion time and throughput.

收稿日期:2020-10-16

基金项目:国家自然科学基金资助项目(61772553;62172441);湖南省研究生创新型项目资助(2020zzts138;CX20200211)

*通信作者,E-mail: dxh@csu.edu.cn

Keywords: edge computing; task decomposition; task scheduling; task association; genetic algorithm; task priority

科技的快速发展与更新迭代,促发了人们对于高并发量、快速计算的需求,并且随之衍生出的网格计算[1]解决方案聚焦于利用各地服务器集中式地解决单任务.然而,现有计算资源已难以支撑不断攀升的互联网计算量及计算需求等.例如,汽车制造业的迅猛发展,汽车已经快速地进入千家万户,并随之给交通系统、交通安全、交通监管、道路规划带来了巨大的挑战.智能交通系统(Intelligent Traffic System,ITS)、车载网络(Vehicular Ad Hoc Networks,VANETs)、车辆网(Internet of Vehicle,IOV)的提出,旨在解决汽车之间通信、汽车与道路设施通信、车辆监控、车流量预测、车辆行为识别等问题,以缓解道路交通问题.如何解决用户对于快捷、便利服务需求.显然,寻找替代方案及开发新技术迫在眉睫[2],旨在研究如何提高计算资源利用率、保障用户服务质量、减少计算资源浪费等.

当前,我国面临高质量发展、数字转型等新需求,提出“加快 5G 网络、数据中心等新型基础设施建设进度”.边缘计算[3]作为关键核心技术之一,在引领驱动“新基建”建设中发挥重要作用.边缘计算是一种基于云计算和并行计算(PC)[4]的新颖计算范式,引起了国内外研究机构及企业的广泛关注及研究.其优势在于能够高效地实现资源利用而忽略各种计算资源底层差异性,以保障计算资源服务可靠性.边缘计算中最重要的技术之一资源分配管理任务,包含资源分配、异构资源管理等,且该任务对边缘计算的能耗、性能、成本等具有较大的影响.非边缘计算场景的传统调度算法以提升资源利用率及总任务完成时间为目标.然而,由于边缘计算场景中存在资源异构、网络环境复杂、用户提交的任务波动性高等特点,因此,调度算法[5]需要同时满足多方面目标.显然,高性能、高效率的任务分解[6-7]及调度算法是保障边缘计算服务及管理水平的关键.

针对不同边缘计算场景及特定任务,任务分解方法及调配策略也是不同的.任务分解按照一定规则分解一个复杂的大任务.这些复杂的大任务将会被分解为多个粒度适宜的、相互独立的子任务,并考虑各子任务间的独立性和制约关系,保证多个对应服务器同时对子任务的协同分析[8]、处理及完成.在这个过程中,也需要考虑任务本身特点、性质、规模等问题.合适的边缘调度策略将最大化地提供合理的系统资源调配,以保证边缘计算模式下任务高效处理、提高资源有效利用.

遵循资源最大化利用原则及边缘计算“廉价按需”宗旨,本文依赖于任务特性进行分解的基础上,建立子任务优先级队列,以最大化实现期望资源按需调度分配准则.此外,通过定义任务关联关系,设计一种任务关联融合的任务分解模型,并讨论任务相似度及信息输入输出间的相互关系对任务分解的影响.通过子任务优先级设计资源分配.

主要贡献总结:

1)结合任务相似度和信息输入输出之间的关联性分析任务关联度.根据任务的相关性将任务进行一定的分解,保证子任务能够并行运行,提高子任务执行效率以及资源利用率.以最小化任务关联性设计任务分解目标函数,采用遗传算法对任务进行优化分解.

2)按照任务优先级设计资源分配方案.利用优先级进行调度能够很大程度上保证任务执行的实时性、高质量以及服务质量.

3)根据 OMNET++工具对 TDAF 算法性能进行仿真,验证结果表明 TDAF 算法性能优于其他 4 种任务调度算法.

1　相关工作

任务的复杂化以及用户对于快捷、便利的服务需求,促使各个企业和研究学者探求性能更加优良的任务分解和任务调度算法[9].以下将从任务分解和任务调度 2 种算法出发,分别介绍 2 个方向已有的研究情况.

1.1 任务分解

任务分解问题自提出以来得到了研究者的广泛关注,已有很多的研究者针对该方向提出了许多研究方案.当子任务是独立的结构时,多个工人被组织成并行协作,并且一个子任务的结果质量只取决于执行该子任务的工人所付出的努力.典型的微任务包括识别照片中的物体、转录录音、研究数据细节和翻译任务.

目前,按照分解的机理特性可以将其分解为 2 类:一类是从任务自身特征出发,采用一种形式化的解决方法对任务以及任务的分解方式进行系统描述,即将一个任务分解问题变化为对一组或者多组信息进行分解的问题.这类方案可以通过求解最优近似解确定任务分解的决策,即启发式分解方案[10-12].另外一类是基于多协作的任务分解方案[13],对任务分解的研究主要集中在 3 个方面:任务相关性的描述、任务流的描述和基于任务分解的方法.在任务流描述中,常用的方法有设计组织矩阵、Petri 网、有向无环图[14](Directed Acyclic Graph, DAG)、任务交互图[15](Task Interaction Graph, TIG)、分层任务网络[16](Hierarchical Task Networks,HTN),每种方法的选择主要是基于任务描述的要求.文献[17]基于 HTN 规划和部分序列因果链(POCL)规划的概念提出了一种新的启发式混合规划方法.文献[18]考虑了扩散任务的分解方案,粒子群算法被用来优化分解模型.

1.2 任务调度

任务调度卸载策略的主要任务是研究用户终端生成的任务是否需要卸载以及任务被调度在何处.把用 1 个边缘服务器无法满足服务质量需求的任务,按照某一种操作分配给多个边缘服务器进行处理的操作过程称为任务调度.如以 Hadoop 为基础实现了先进先出调度程序,容量调度程序,文件调度程序等经典调度算法,另外还有最小最小调度算法(Minimum-Minimum completion time,Min-Min)、最大最小调度算法(Maximum-Minimum completion time,Max-Min)算法.

在 MEC 领域,一个重要的问题为是否以及如何将任务分配到移动边缘的合适位置,以降低运行成本且提高 QoS.一些研究人员建议在边缘网络设施(例如基站和路由器),并将任务转移到边缘网络设施上的边缘云[19-22].Kwak 等[19]研究了动态资源分配和任务卸载问题,提出了在给定时滞约束条件下优化能耗的 Lyapunov 优化框架;文献[20]研究了多个用户任务卸载一个任务的问题,把该问题表述成了 TDMA 下凸优化的问题和 OFDMA 下混合整数规划问题;Liu 等[21]讨论了能耗与延迟的联合优化问题,提出了一种基于马尔可夫决策过程的能量约束延迟最优任务卸载策略;Jiang 等[22]分析了时延和能耗之间的权衡,并进一步提出了一种基于 Lyapunov 优化框架的在线算法来联合优化时延和能耗.然而,在实践中,由于技术和经济原因,部署和维护 MEC 服务器的成本往往很高.为了克服边缘云的缺点,一些研究者提出了一种互补的方法,将任务转移到资源空闲的附近移动设备上[23].为了实现这一点,移动设施需要 D2D 进行协作和资源共享.然而,以往的研究没有考虑到任务资源需求的异质性,而在实践中,任务对资源的需求往往是异构的、多维的.例如,AR 任务需要通信资源上传实时视频数据、计算资源处理视频数据.为此,研究者进一步提出将一个任务根据其资源需求划分为多个微任务,并将不同的微任务适当地分配给不同的移动设备[24-26].具体来说,Chen 等人[24]通过 D2D 协作构建了一个激励感知资源共享框架,其中每个任务可以使用多个移动设备来完成不同的任务;Tang 等人[25]提出了移动设备间多维资源共享的一般框架,其中每个任务被划分为 3 个微任务(即不同的微任务可以卸载到不同的设备上);Pan 等[26]建立了更一般的任务模型,将每个任务划分为 5 个微任务(即数据下载、处理、上传,2 个移动设备间数据通信的 D2D 微任务).显然,通过这样的任务划分,可以更加灵活有效地分配任务.

2 算法模型

每一个任务在进行处理时,对于服务质量有各自不同的需求.有些用户要求任务必须在指定时间内完成,有些用户要求任务完成成本必须低廉,有些用户要求任务完成不仅要成本低廉还要保证执行时间.边缘环境下的任务分解和任务调度问题不仅要解决任务执行的高效性、并行性,还要在任务执行期间权衡网

络资源消耗(能耗、带宽等).基于此本文提出一种边缘计算中基于任务分解的任务调度算法.将一个复杂任务分解为 n 个独立的子任务.设计、计算子任务优先级,按照子任务优先级进行资源调度分配,将每一个子任务按照其自身特性分配到合理的资源进行执行.

2.1　任务分解模块

已知复杂任务 Task,将 Task 分解为 n 个子任务.换言之任务 Task 可以表示为由 n 个被分解后的子任务组成的集合,即 $\text{Task}=\{\text{task}_1,\text{task}_2,\cdots,\text{task}_n\}$.任务分解过程可以用层次的树型结构表示(任务树,如图 1 所示).该任务树中的每一个节点都可以表示为一个有限的集合,根节点是总任务,除了根节点其他节点都可以按照某种操作进行分解.

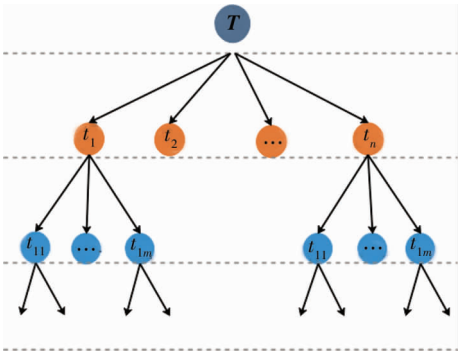

图 1　任务分解树

2.1.1　子任务关联关系

分解后的子任务之间相互影响,存在关联关系.依据分解后的子任务之间是否存在依赖,可以将子任务间关联关系分为 3 种:独立型、依赖型和耦合型,如图 2 所示.

1)独立型.子任务之间没有依赖关系,相互独立、不相互影响、支持.子任务间独立型可以表示为 Ind(子任务 1,子任务 2).

2)依赖型.子任务需要其他子任务依赖、支持,即子任务 1 需要子任务 2 提供支持才可以进行.子任务间依赖可以表示为 Dep(子任务 1,子任务 2).

3)耦合型.子任务执行相互依赖、支持,即子任务 1 的执行需要子任务 2 提供支持才可以进行,子任务 2 的执行需要子任务 1 提供支持才可以进行.子任务间耦合可以表示为 Cou(子任务 1,子任务 2).

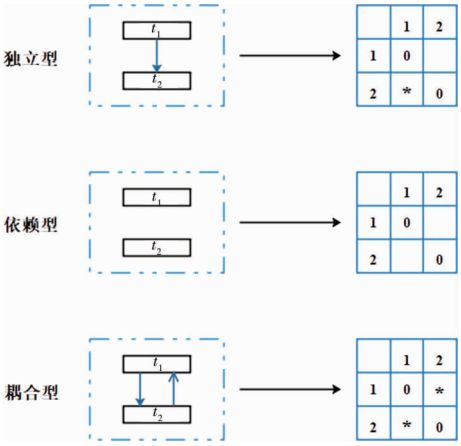

图 2　任务关联关系

定义 1:子任务间关联关系矩阵(Task Association Matrix,**TAM**).假设分解后的 n 个子任务关系可以用一个 $n\times n$ 的子任务关联关系矩阵表示.**TAM** 矩阵中的任意一列、任意一行代表一个子任务.则 **TAM** 可以表示为

$$\textbf{TAM}=[I_{ij}]_{n\times n}. \tag{1}$$

式中:I_{ij} 为子任务 t_i 给子任务 t_i 的作用强度.此处,定义子任务自身关系为 0,即 $I_{ii}=0$.

定义 2:子任务相似度.**TAM** 矩阵中的第 i 行是其他子任务对子任务 i 的作用强度.第 j 列表示其他子任务对子任务 j 的作用强度.通过使用 Jaccard 相似系数可以得到 **TAM** 矩阵中行与行、列与列之间的相似性.

$$\text{col}_{ij}=\frac{\sum_{r=1,r\neq i,j}^{n}\min(I_{ir},I_{ir})}{\sum_{r=1,r\neq i,j}^{n}\max(I_{ir},I_{jr})},\quad i,j\in[1,n]; \tag{2}$$

$$\mathrm{row}_{ij}=\frac{\sum_{r=1,r\neq i,j}^{n}\min(I_{ri},I_{rj})}{\sum_{r=1,r\neq i,j}^{n}\max(I_{ri},I_{rj})},\quad i,j\in[1,n]. \tag{3}$$

故而子任务间相似度为

$$\mathrm{TS}_{ij}=\mathrm{row}_{ij}+\mathrm{col}_{ij}. \tag{4}$$

定义 3:子任务信息输入输出关联度.对于存在信息输入输出的子任务,可以通过子任务对于接收到其他子任务的信息满足程度定义子任务间信息输入输出关联度.例如,子任务 i 与子任务 j 之间存在信息输入输出关系,子任务 i 的信息输出是子任务 j 的信息输入,子任务 i 的输出在很大程度上会影响子任务 j 的输出.定义子任务 i 的信息输出为 MO^i, 子任务 j 的信息输出为 MI^j, 则 MI^j 对于 MO^i 的满足程度可以表示为 $S'_{ij}=\xi'_{ij}(\mathrm{MO}^i,\mathrm{MI}^j)$. 故而可以得到子任务 i 和任务 j 间的信息输入输出关联度:

$$\delta'_{\mathrm{AB}}=\sum_{i=1}\sum_{j=1}\xi'_{ij}(\mathrm{MO}^i,\mathrm{MI}^j). \tag{5}$$

式中:$\xi'_{ij}(\cdot)$ 是信息满足度函数并且是为关于 MO^i 的递增函数.

定义 4:子任务关联度.子任务之间关联有直接关联(子任务之间信息输入输出关联度)和间接关联(子任务相似度)2 个方面,子任务关联度可表示为

$$\mathrm{TC}_{ij}=\delta'_{\mathrm{AB}}+\mathrm{TS}_{ij}. \tag{6}$$

2.1.2 *遗传算法优化任务分解*

对于一个需要被分解为 m 个子任务的复杂任务,需要在 **TAM** 矩阵中寻找 $m-1$ 个分割点将其进行分解,从而得到 m 个子任务.为了使分解后的子任务能够尽量相互独立,以最小化任务关联性设计任务分解目标函数,采用遗传算法对任务进行优化分解.

$$\begin{aligned}&\min\quad T=\sum_{k=1}^{m}\sum_{i=f_k}^{l_k}\sum_{j=f_k}^{l_k}TC_{ij}-\sum_{k=1}^{m-1}\sum_{i=f_k-1}^{l_k}\sum_{j=f_k}^{l_k-1}TC_{ij}\\&s.t.\quad r_{\min}\leqslant l_k-f_k+1\leqslant r_{\max},\quad k\in[1,m].\end{aligned} \tag{7}$$

式中:f_k,l_k 为第一个子任务和最后一个子任务;$r_{\min}$,$r_{\max}$ 为被分解的子任务最小数目和最大数目.

运用遗传算法求解式(7).将式(7)最分解问题解转化为遗传算法搜寻解问题.利用二进制编码方式对任务分解的 $m-1$ 个分解点进行编码,该编码可以表示为 $\mathrm{en}=[x_1,x_2,\cdots,x_n]$,即染色体编码.其中,该编码中的每一个基因位 x 表示:

$$x_i=\begin{cases}0 & \textbf{It's a split point}\\1 & \textbf{It's not a split point}\end{cases},\ \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x_i}=\mathrm{m}-1. \tag{8}$$

式中:$x_i=1$ 表示该基因位置为分割点;反之 $x_i=0$ 该基因位置不是分割点.利用遗传算法,初始化种群可以用以下步骤完成:

1)任意取二进制编码 $\mathrm{en}=[x_1,x_2,\cdots,x_n]$ 中一个元素位置,并设置其值为 1,其他的元素值为 0.对于基因位元素值同为 1 的位置,这两位置间的距离必须满足约束:$r_{\min}\leqslant d\leqslant r_{\max}$.

2)重复执行 1,直到生成一个初始化的种群.

选择算子:运用赌轮的方法进行选择,染色体的适应值大小与其被选择的概率成正比.染色体适应值和染色体总适应值的比重可以看成是一个圆盘,该圆盘的面积值为 1.转动指针,指针停止后所指的位置将被复制到后代,过程如下:

1) 染色体适应值

$$\mathrm{Fit}\ (i)=G(\mathrm{en}_i),i=1,2,\cdots,\mathrm{psize}. \tag{9}$$

2) 种群适应值

$$F=\sum_{i=1}^{\mathrm{psize}}\mathrm{Fit}\ (i). \tag{10}$$

3) 每一个染色体被选择的概率

$$S_i=\frac{\mathrm{Fit}\ (i)}{F}. \tag{11}$$

4) 每一个染色体被选择的被选择的累计概率:

$$C_i = \sum_{i=1}^{i} S_i. \tag{12}$$

5) 循环生成介于[0,1]的随机数 h.如果 $r<C_i$,对应的染色体 i 复制后代直到生成种群.该种群大小为 psize .

交叉算子:对于染色体交叉点 cpoint 的选取需要按照位于交叉点两侧的基因位元素数值为 1 的个数相同的约束条件进行,即 $\sum_{i=1}^{\text{cpoint}} x_i = \sum_{i=1}^{\text{cpoint}} y_i$.

变异算子:随机选择染色体编码 en 中值为 1 的位置和值为 0 的位置,将这 2 个位置进行交换. 对于进行变异后的染色体而得到的新染色体需要满足 $r_{\min} \leqslant d \leqslant r_{\max}$.

2.2　任务调度模块

一个任务能否顺利并行,主要取决于所涉及的任务执行过程是否合理,该过程包含了对任务进行分解和对任务进行调度、分配等.并行设计中任务调度、分配就是将已经规划好的子任务分配给合理的资源执行,从而缩短整个任务执行周期.本小节针对上一节中任务分解的结果,按照不同子任务的优先级设计合理的资源调度、分配方案.

为了使得任务调度问题具有普适性,我们假设上一节已经将一复杂任务分解为了 n 个合理的、独立的子任务,并且边缘计算环境中有 $g(g<n)$ 个有效的计算执行资源可以执行 n 个已分解的子任务.任务优先级对于任务调度具有很大程度影响.本节将从以下 4 个方面设计考虑子任务的优先级.

1) 子任务执行完成时间.子任务执行完成时间应该由已经执行完成的其他子任务最早执行完成时间决定.因为某一子任务执行完成时间越长,如果在执行该子任务过程中执行出现失败,则需要再次执行该子任务,那么该过程将会增加整个任务执行完成时间并且加大资源的消耗.

2) 子任务后继子任务数目.一个子任务后继子任务数目越多,该子任务在整个任务中所占的比重越重要,后继子任务数目记为.

3) 子任务影响力.子任务 i 执行完成时间占整个任务执行完成时间的比值:

$$\eta = \frac{e_i}{e_n}. \tag{13}$$

4) 资源利用率.子任务 k 在执行时其在某一段时间内使用某一个计算资源的频率,即子任务 k 对于某一个计算资源的使用频率.

$$\mu = \max\left[\frac{1}{e_i}\sum_{i=1}^{R_k}\sum_{k=1}^{T_k}\frac{\sigma_k^j}{R_k}\right]. \tag{14}$$

式中:σ_k^j 为子任务 k 使用计算资源 j 的频率.从以上分析可以得到子任务优先级为

$$p_i = \alpha_1 t_i + \alpha_2 \gamma_i + \alpha_3 \eta_i + \alpha_4 \mu_i. \tag{15}$$

式中:$\alpha_1,\alpha_2,\alpha_3,\alpha_4$ 分别为影响子任务优先级 4 个方面的影响因子权重系数,该权重系数大小表示了 4 个方面对于子任务优先级的影响程度大小.图 3 是任务优先级状态图,同一列的任务具有相同的优先级.

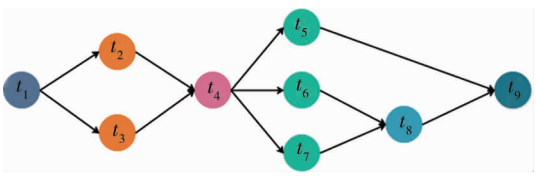

图 3　任务优先级

按照任务优先级关系将 n 个子任务合理地分配到 m 个计算资源来执行.子任务执行状态集合用 $\boldsymbol{T}=[t_1,t_2,\cdots,t_n]$ 表示,$t_i=1$ 表示子任务 i 正在执行,$t_i=0$ 表示子任务 i 正在等待执行,$t_i=2$ 表示子任务完成.$\boldsymbol{R}=[R_1,R_2,\cdots,R_m]$ 表示 m 个计算资源集合.R_i 表示子任务 i 执行所需的计算资源大小.$R_i^{\max}$ 表示子任务 i 所需的最大计算资源大小,当子任务执行所需的计算资源大小超高边缘服务器所能提供的最大计算

资源大小时对于子任务的执行速度的加速没有多大作用.$R_i^{\min}$ 表示子任务 i 所需最小计算资源大小,因为当计算资源过小时子任务无法执行.故而子任务 i 对计算资源的需求可表示为 $R_i = \{R_i^{\min}, R_i^{\max}\}$.

3 仿真与结果

在离散事件模拟器 OMNET++环境中来评估 TDAF 算法的性能.任意给出 100 个任务,分每组 10 个任务进行 10 组仿真实验,创建 10 个边缘服务器节点作为计算资源.实验选择 3 种算法进行对比验证本文算法的性能,分别从完成时间和吞吐率 2 个方面进行了评估.3 种算法分别是 Max-min 算法、TBSTD 算法和 Max-int 算法.Max-min 算法和 Max-int 算法是经典的任务调度算法,这 2 种算法对任务不经过任何处理直接进行调度分配,与 Max-min 和 Max-int 算法的对比实验能够很好地反映出任务分解对于任务处理在时间和空间上的优势.TBSTD 算法采用“时间均衡”的思想设计任务调度,该算法仅仅从任务处理时间出发却没有考虑任务自身大小等因素.

3.1 任务执行完成时间

设计 2 组实验对 Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法任务执行完成时间进行验证.一组实验通过考虑不同任务数目个数对于任务执行完成时间的影响.另一组实验通过设计不同任务大小验证任务大小对于任务执行完成时间影响.

实验一:设置任务数目:100,500,1 000,4 000,7 000,10 000.不同任务数目所对应的任务大小:260,330,450,360,500,500,带宽:500,600,700,700,900,1 000.任务执行完成时间随任务数目大小的变化如图 4 所示.从图 4 中可以看出随着任务数目的不断增加,Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法的任务执行完成时间逐渐增加.任务数目为 100,即任务数目较小时,Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法任务执行完成时间几乎相同没有很大的差距.由图 4 可知,TDAF 算法、Max-min 算法、TBSTD 算法和 Max-int 算法的任务执行完成时间分别为 2.476,2.793 ,2.557,2.679 s.显而易见,在相同任务数目的情况下,TDAF 算法的任务执行完成时间最低.这是由于 Max-min 算法和 Max-int 算法是在对任务不经过处理的基础上直接进行任务执行,导致任务粒度过大,无法充分的利用系统资源,导致任务执行完成时间增大.TBSTD 算法和 TDAF 算法任务执行完成时间相差较小,TBSTD 算法通过时间均衡的策略将任务分解,没有考虑到任务优先级导致任务执行拥塞,任务执行完成时间加大.TDAF 算法是在将任务进行合理的分解基础上再对任务进行调度、分配,这样可以在很大程度中利用空闲的资源、提高资源利用率,降低任务执行完成时间.

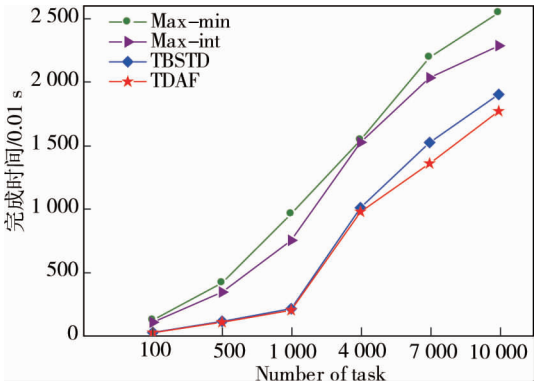

图 4 任务数目对完成时间影响

实验二:共设有 10 个任务,每一个任务按照编号进行标注 0,1,2,3,4,5,6,7,8,9.不同任务编号所对应的任务大小:270,550,860,900,1 500,2 500,3 000,4 000,5 000,7 000,带宽:900,1 000,1 200,1 500,1 900,2 300,3 000,3 200,4 300,4 000.任务执行完成时间随任务大小的变化如图 5 所示.从图 5 中可以看出随着任务大小的不断增加,Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法的任务执行完成时间逐渐增加,TBSTD 算法和 TDAF 算法增加趋势相对较平缓.任务大小较小时,Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法任务执行完成时

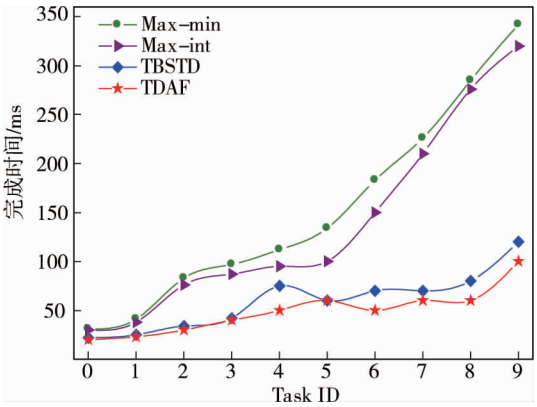

图 5 任务大小对完成时间影响

间几乎相同差距不大.由图 5 可知,TDAF 算法、Max-min 算法、TBSTD 算法和 Max-int 算法的任务执行完成时间分别为 4.376,16.240 ,5.572 ,15.860 s.显而易见,TDAF 算法的任务执行完成时间最低.TDAF 算法将分解后的子任务按照任务的优先级设定资源分配方案,该方法能够很好地利用计算资源,从而达到降低任务执行完成时间的目的.

3.2　吞吐率

图 6 所示是系统吞吐率随任务数目的变化.从图 6 中可知随着任务数目的不断增加,Max-min 算法、TBSTD 算法、Max-int 算法和 TDAF 算法的系统吞吐率呈现起伏波动状态.Max-min 和 Max-int 算法在进行任务处理过程中未对任务进行分解,因而任务粒度较大,不能充分利用空闲可用资源,从而导致系统吞吐率较低.TBSTD 算法仅仅通过考虑时间均衡策略对任务进行分解,TDAF 算法从任务本身特性出发对于任务进行分解, 由分解后的子任务寻找对应的资源进行处理,充分利用了资源,有利于提高系统吞吐率.

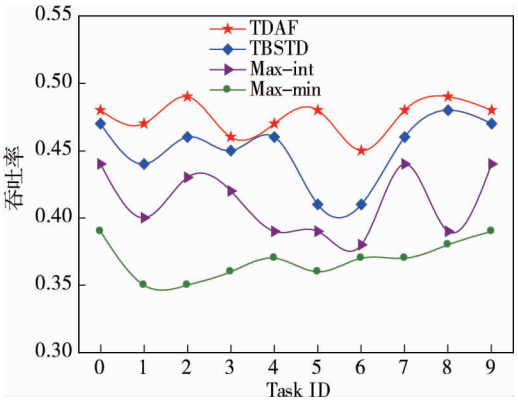

图 6　吞吐率

4　结论

1)提出了一种基于任务分解的任务调度算法,从中可依据不同的评价指标比较快的寻求到相应的最优方案.实验表明 TDAF 算法比已有的 Max-min 算法、TBSTD 算法、Max-int 算法求解效率有显著提高.

2)针对任务间的关联性和子任务间的相关性设计任务分解目标函数,遗传算法可以灵活的寻找到不同计算任务的最优任务分解方案.优先级的设计能够有效地提高任务处理速度.

3)该算法对于高效地处理大规模计算密集型和时间敏感性应用任务具有重要的意义.

参考文献:

[1] Mahmud R, Buyya R. Grid Computing [M]// Fog and Edge Computing, 2019.

[2] Hayes B. Cloud computing [J]. Communications of the Acm, 2008, 51(7): 9-11.

[3] Shi W, Cao J, Zhang Q, et al. Edge Computing: Vision and Challenges [J]. Internet of Things Journal, IEEE, 2016, 3(5): 637-646.

[4] Eddelbuettel D. Parallel computing with R: A brief review [J]. Wiley Interdiplinary Reviews: Computational Stats, 2020, 13(2): e1515.

[5] Fu Z, Tang Z, Yang L, et al. An Optimal Locality-Aware Task Scheduling Algorithm Based on Bipartite Graph Modelling for Spark Applications [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(10): 2406-2420.

[6] Feng G , Luo W , Li C. A Study of Cloud Computing Scheduling Algorithm Based on Task Decomposition [C]//2016 3rd International Conference on Engineering Technology and Application(ICETA 2016), 2016.

[7] Huo Y H, Chang H M, Cao Y. Task Decomposition Method Based on Multiple Constraint Relations [J]. Radio Communications Technology, 2016,42(1):35-37.

[8] Sun C, Liu W, Dong L. Reinforcement Learning With Task Decomposition for Cooperative Multiagent Systems [J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 32(5): 2054-2065.

[9] 魏妮妮, 宋翌. 一种基于任务分解的时间均衡调度算法[J]. 河北科技大学学报. 2013(6):559-564.

[10] Su X H, Zhang H L. Cloud computing heuristic task decomposition algorithm in the improvement [J]. Electronic Design Engineering, 2012,20(23): 47-49.

[11] Bao B F , Yang Y , Fei L I , et al. Decomposition model in product customization collaborative development task[J]. Computer Integrated Manufacturing Systems, 2014, 20(7):1537-1545.

[12] 李玉, 党德玉. 一种改进的基于多 Agent 协作的任务分解算法[J]. 东北电力大学学报(自然科学版), 2008(4): 48-51.

[13] 高斐, 邱雪松, 高志鹏, 等. 基于多代理协作的 IT 复杂应用管理任务分解算法[J]. 软件学报, 2011, 22(9): 2049-2058.

[14] Bris R. Parallel simulation algorithm for maintenance optimization based on directed Acyclic Graph [J]. Reliability Engineering & System Safety, 2008, 93(6): 874-884.

[15] 杜晓丽, 蒋昌俊, 丁志军,等. 异构环境下基于任务交互图的调度算法[J].同济大学学报(自然科学版), 2007, 35(3):406-411.

[16] Georgievski I, Aiello M. HTN planning: Overview, comparison, and beyond [J]. Artificial Intelligence, 2015, 222(5): 124-156.

[17] Bercher P, Keen S, Biundo S. Hybrid Planning Heuristics Based on Task Decomposition Graphs[J]. Seventh Annual Symposium on Combinatorial Search, 2014: 35-43.

[18] An B, Liao W, Guo Y, et al. Study on Extended Manufacturing Task Planning Method Based on Correlation [J]. China Mechanical Engineering, 2011, 22(23): 2839-2844.

[19] Kwak J, Kim Y, Lee J, et al. DREAM: Dynamic Resource and Task Allocation for Energy Minimization in Mobile Cloud Systems [J]. IEEE Journal on Selected Areas in Communications, 2015, 33(12): 2510-2523.

[20] You C, Huang K, Chae H, et al. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading[J]. IEEE Transactions on Wireless Communications, 2017, 16(3):1397-1411.

[21] Liu J, Mao Y, Zhang J, et al. Delay-Optimal Computation Task Scheduling for Mobile-Edge Computing Systems[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2016: 1604.07525.

[22] Jiang Z, Mao S. Energy Delay Trade-Off in Cloud Offloading for Mutli-Core Mobile Devices[C]//2015 IEEE Global Communications Conference (GLOBECOM), 2015: 1-6.

[23] Hong X, Liang L, Jie X, et al. Joint Task Assignment and Resource Allocation for D2D-Enabled Mobile-Edge Computing [C]//IEEE Transactions on Communications, 2019, 67(6): 4193-4207.

[24] Chen X, Pu L, Gao L, et al. Exploiting massive D2D collaboration for energy-efficient mobile edge computing[J]. IEEE Wireless Communications,2017, 24(4): 64-71.

[25] Tang M, Gao L, Huang J. Enabling Edge Cooperation in Tactile Internet via 3C Resource Sharing [J]. IEEE Journal on Selected Areas in Communications, 2018, 36(11): 2444-2454.

[26] Pan Y , Gao L , Luo J , et al. A Multi-Dimensional Resource Crowdsourcing Framework for Mobile Edge Computing [C]// ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020: 1-7.