MC2ETS:移动云计算中一种能效任务调度算法

叶符明 李雯婷 王 颖
(贵州商学院计算机与信息工程学院 贵阳 550014)

摘 要 移动云计算可以使执行于移动设备上的任务迁移至云端执行,达到降低移动设备能耗、提高任务执行效率的目的。文中研究了移动云计算中DAG模型的任务调度问题,为了解决传统调度算法缺乏对任务完成时间和移动设备能耗的同步优化问题,提出了一种移动云计算的能效任务调度算法MC2ETS(Energy-efficient Tasks Scheduling of Mobile Cloud Computing)。该算法主要包括3个步骤:1)以最小化应用完成时间为目标进行初始调度;2)在满足应用完成时间约束的同时,以最小化能耗为目标进行任务调度迁移;3)通过提出的DVFS(Dynamic Voltage/Frequency Scale)算法进一步降低能耗。通过具体的实例验证了算法的可行性,并分析了算法的时间复杂度。最后,通过与基准算法的系统性实验对比分析,证明了算法在多数情况下可以在调度时间指标与移动设备能耗间实现均衡优化。

关键词 移动云计算,能效,任务调度,任务迁移,均衡优化
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.06.020

MC2ETS:An Energy-efficient Tasks Scheduling Algorithm in Mobile Cloud Computing

YE Fu-ming LI Wen-ting WANG Ying
(School of Computer and Information Engineering,Guizhou University of Commerce,Guiyang 550014,China)

Abstract Mobile cloud computing can migrate the tasks scheduled on mobile devices to cloud,which can reduce the energy consumption of mobile device and improve the tasks execution efficiency. Tasks scheduling problem with Directed Acyclic Graph (DAG) model in mobile cloud computing was studied. Traditional methods for scheduling tasks usually are short of optimizing synchronous both tasks completion time and energy consumption of mobile device,an energy-efficient tasks scheduling algorithm of mobile cloud computing (MC2ETS) was presented in this paper. The algorithm consists of three steps. Firstly,the initial scheduling is carried out to minimize the application completion time. Then the task scheduling migration is conducted based on minimizing the energy consumption,while satisfying the constraint of application completion time. At last,through DVFS(Dynamic Voltage/Frequency Scale) algorithm,the energy consumption is reduced further. The feasibility of the proposed algorithm was verified through the specific example,and the time complexity of the proposed algorithm was analyzed. Finally,through the systemic experimental analysis compared with the baseline algorithms,this paper proved that the proposed algorithm can achieve the trade-off optimization between the scheduling time index and the energy consumption of mobile device in most cases.

Keywords Mobile cloud computing,Energy-efficient,Tasks scheduling,Tasks migration,Trade-off optimization

1 引言

移动设备(如智能手机或笔记本电脑等)已经成为当今物物互联型社会中主要的计算平台。然而,尽管该类移动设备以及无线通信技术发展迅猛,由于受电池容量、存储容量和计算能力等多方面的限制,与传统的功能强大的高性能服务器计算资源相比,移动设备的计算能力仍比较有限[1]。尤其在执行计算密集型应用任务时,计算能力和能耗的限制都将导致用户无法得到服务质量的保证。

云计算依靠其按需服务、普适的网络访问以及区域化的独立资源池的独特优势形成了新一代的高性能计算模式,通过互联网,云计算的服务提供者可以向用户提供强大的计算和存储资源。而在诸如3G,Wi-Fi和4G等无线通信技术的帮助下,移动云计算(Mobile Cloud Computing,MCC)应运而生[2-3],它提供了基于云端资源共享和增强移动计算两者统一的平台,可以允许将计算任务、数据存储等应用任务的处理外包上传至资源不受限的云端执行,以提高可靠性,降低移动设备的能耗[4-5]。

然而,目前移动云计算环境中的任务调度多集中于优化任务完成时间(获得更高的性能)或优化移动设备端的能耗

到稿日期:2018-01-29 返修日期:2018-04-30 本文受贵州省普通高等学校工程研究中心基金项目(黔教合KY字[016]),贵州省教育厅青年科技人才成长项目(黔教合KY字[237])资助。

叶符明(1978—),女,硕士,副教授,主要研究领域为大数据、云计算;李雯婷(1984—),女,博士,副教授,主要研究领域为大数据分析,E-mail:wendy-workmail@163.com(通信作者);王 颖(1982—),女,硕士,讲师,主要研究领域为计算机应用。

(延长移动设备的电池寿命),缺少将完成时间和执行能耗同步优化并考虑某些 QoS(Quality of Service)约束情况下的调度算法。为了解决该问题,本文提出了一种新的移动云计算任务调度算法,该算法可以在满足完成时间约束的同时最小化移动设备的能耗,并重点解决以下问题:1)决定哪些应用任务需要外包至云端;2)决定剩余任务在移动设备内核上的映射;3)在满足任务顺序依赖的同时,决定异构内核和无线传输信道(远程云端执行)上的任务调度,最小化移动设备的能耗。

2 相关研究工作概述

在优化任务完成时间的相关工作中,文献[6]在异构处理器环境中,通过设置任务优先级的方法实现了高效的任务调度。文献[7]利用一种两阶段式方法优化任务调度,第一阶段先利用快速确定算法进行任务初始调度,然后在第二阶段中进行不断迭代,改进初始调度方案。文献[8]利用增量贪心算法进行任务调度优化,该算法可以自适应地实现移动交互感知应用进行任务卸载和并行执行,进而缩短应用的执行时间。文献[9]则对应用任务的分配问题进行了优化,通过在移动设备与云端间实现更大的吞吐量来改进应用任务的调度效率。

在优化移动设备能耗的相关工作中,文献[10]将具有相互依赖关系的工作流式应用任务调度优化问题形式化为最大流/最小分割问题,通过优化在移动设备与云端中执行的每个子任务的形式进行能耗优化。文献[11]设计了一种快速混合式多站式任务调度决策算法,在考虑应用规模的情况下可以得到执行时间与能耗同步优化的近似最优解。文献[12]优化了具有周期性特征的任务调度问题,为了优化调度能耗,算法假设任务周期足够大,通过缩短任务调度间隔时间达到优化能耗的目标。文献[13]提出了一种基于云协助的计算卸载机制,可以实现计算密集型应用的执行能耗优化。文献[14]提出了一种上下文感知的计算卸载框架,并利用一种估算模型对最优的云资源选择进行决策。文献[15]对文献[6]的工作进行了改进,对异构处理器环境中的能耗和效率问题进行了分析,但最终的调度并不能确保问题定义中预先设定的完成时间约束。

时间优先和能耗优化本身是两个相互冲突的目标[16],若要缩短任务调度完成时间,势必需要选择性能更好的处理器资源,但资源性能越高,能耗也相对更高。而为了降低能耗,势必会选择低功率的处理器执行任务调度,这又会延长完成时间,尤其是在具有完成时间约束的情况下,这将直接导致服务质量的下降。为了解决这一问题,本文设计一种移动云计算中的任务能效调度算法,该算法将同时优化任务完成时间和移动设备能耗,在满足完成时间最大约束的同时实现调度效率与能耗优化的均衡。

3 移动云计算 MCC 的任务调度模型

3.1 MCC 中的应用任务

一个 MCC 应用可表示为一个有向无循环任务图 DAG (Directed Acycle Graph)[17] $G=(V,E)$,其中,节点 $v_i \in V$ 表示一个任务,一条有向边 $e(v_i,v_j) \in E$ 表示任务间的一种顺序约束,即任务 v_i 需要在任务 v_j 开始执行前完成。DAG 中的任务总数为 N,对于给定的任务 DAG,若任务没有父任务,则称之为入口任务(entry task);若任务没有子任务,则称之为出口任务(exit task)。如图 1 所示的任务 DAG,任务 v_1 为入口任务,v_{10} 为出口任务。对于每个任务 v_i,若该任务被外包卸载至云端执行,则定义 $data_i$ 为任务计算量与要求上传至云端的输入数据量之和,$data_i'$ 为需从云端下载的数据量。

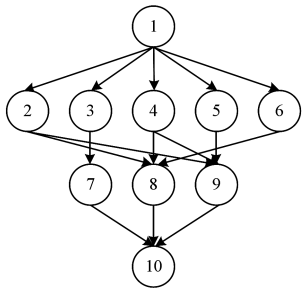

图 1 任务 DAG

Fig. 1 Task DAG

3.2 MCC 环境

MCC 环境中的移动设备可以通过无线通信方式访问云端资源。假设移动设备处理器内拥有 K 个异构内核,内核中均配置了动态电压与频率调整(DVFS),使得每个内核可运行在 M 个不同的频率等级(对应于 M 个供电电压等级)下。令第 k 个内核的最大运行频率为 $f_k^{\max}$,且拥有 M 个频率调整因子,即 $\alpha_{k,1}<\alpha_{k,2}<\cdots<\alpha_{k,M}=1$。因此,第 k 个内核的实际运行频率为 $f_k=\alpha_{k,m}\times f_k^{\max}$。第 k 个内核的功耗 P_k 是频率 f_k 的超线性函数,可表示为:$P_k=\alpha_k\times(f_k)^{\gamma_k}$,其中,$2\leqslant\gamma_k\leqslant 3$。$\alpha_k$ 和 γ_k 的取值在不同内核上也不同。

一个任务可在移动设备的内核上执行(即局部调度),也可在远程云端执行(即云端调度)。若任务 v_i 被外包至云端执行,则其执行会顺序经历以下 3 个阶段:1)无线射频(Radio Frequency,RF)发送阶段;2)云端计算阶段;3)RF 接收阶段。在 RF 发送阶段,移动设备将任务的计算量及输入数据量通过无线发送信道(wireless sending channel)发送至云端;在云端计算阶段,任务在云端资源上执行;在 RF 接收阶段,移动设备通过无线接收信道(wireless receiving channel)接收任务的输出数据量。一旦云端完成任务,就需要将任务输出数据送回移动设备。令 R^s 表示无线发送信道的数据发送速率,R^r 表示无线接收信道的数据接收速率,P^s 表示发送数据至云端时移动设备中 RF 组件的功耗等级。移动设备中 RF 组件在接收数据时的功耗相比发送数据可忽略不计。

假设移动设备内核和发送信道一次仅能同时处理或发送一个任务,而云端可并行执行不存在顺序依赖的多个任务。

3.3 MCC 中的任务依赖要求

令 $T_{i,k}^l$ 为任务 v_i 在移动设备第 k 个内核上的执行时间,符号"l"表示局部执行(local execution)。$T_{i,k}^l$ 与运行频率 f_k 成反比。令 T_i^c 为任务 v_i 在云端的执行时间,符号"c"表示在云端执行(cloud execution)。令 T_i^s 为发送任务 v_i 至云端的时间,且:

$$T_i^s=\frac{data_i}{R^s} \tag{1}$$

令 T_i^r 为从云端接收任务 v_i 输出数据的时间,且:

$$T_i^r=\frac{data_i'}{R^r} \tag{2}$$

对于已在局部内核或云端调度的任务 v_j，令 FT_j^l 为任务 v_j 在局部内核上的完成时间，FT_j^{ws} 为任务 v_j 在无线发送信道上（即任务被完全上传至云端）的完成时间，FT_j^c 为任务 v_j 在云端的完成时间，FT_j^{wr} 为任务 v_j 在无线接收信道上（即移动设备完全接收云端的任务输出数据）的完成时间。若任务在局部调度，则 $FT_j^{ws}=FT_j^c=FT_j^{wr}=0$；否则（即任务外包至云端）$FT_j^l=0$。移动设备仅能在局部内核和无线信道上调度任务，而云端负责调度外包上传的任务并送回输出数据，但移动设备可依据 T_j^s 和 T_j^r 预计云端的任务执行对应的时间 FT_j^c 和 FT_j^{wr}。

1）局部调度

调度任务 v_i 前，该任务的所有直接前驱父任务必须确保已经被调度。假设任务 v_i 被调度至一个局部内核，则任务 v_i 的就绪时间 RT_i^l 可计算为：

$$RT_i^l=\max_{v_j\in pred(v_i)}\{FT_j^l,FT_j^{wr}\} \tag{3}$$

其中，$pred(v_i)$ 为任务 v_i 的直接前驱任务集。v_i 的就绪时间 RT_i^l 即为其所有前驱任务完成的最早时间，且其结果对于任务 v_i 是可用的。

①若任务 v_j（任务 v_i 的一个直接前驱任务）已在局部调度，则 $\max\{FT_j^l,FT_j^{wr}\}=FT_j^l$。此时有 $RT_i^l\geqslant FT_j^l$，这表明 v_i 仅在 v_j 局部执行后才可以开始在一个局部内核上执行。

②若任务 v_j（任务 v_i 的一个直接前驱任务）已外包至云端执行，则 $\max\{FT_j^l,FT^{wr}j\}=FT_j^{wr}$。此时有 $RT_i^l\geqslant FT_j^{wr}$，这表明 v_i 仅在移动设备上通过无线接收信道从云端完全接收 v_j 的数据后才可以开始在一个局部内核上执行。

若任务在局部内核上调度，则任务 v_i 仅在其就绪时间 RT_i^l 时或之后才能开始执行，此时必须保证任务间的依赖关系。然而，由于内核可能同时执行其他任务，故并不能保证任务 v_i 在其就绪时间点上准确开始执行。

2）云端调度

若任务 v_i 外包至云端执行，令 RT_i^{ws} 为 v_i 在无线发送信道上的就绪时间，且：

$$RT_i^{ws}=\max_{v_j\in pred(v_i)}\{FT_j^l,FT_j^{ws}\} \tag{4}$$

其中，RT_i^{ws} 表示在确保任务间相互约束条件下，任务 v_i 在无线发送信道上调度的最早开始时间。

①若任务 v_j（任务 v_i 的直接前驱任务）已在局部调度，则 $\max\{FT_j^l,FT_j^{wr}\}=FT_j^l$。此时 $RT_i^{ws}\geqslant FT_j^l$，这表明移动设备仅在 v_j 局部调度完后才可以通过无线信道开始发送任务 v_i。

②若任务 v_j（任务 v_i 的直接前驱任务）已外包至云端，则 $\max\{FT_j^l,FT_j^{ws}\}=FT_j^{ws}$。此时 $RT_i^{ws}\geqslant FT_j^{ws}$，这表明移动设备仅在其将 v_j 外包上传至云端后才开始通过无线信道发送任务 v_i。

令 RT_i^c 表示云端中任务 v_i 的就绪时间，计算式为：

$$RT_i^c=\max\{FT_i^{ws},\max_{v_j\in pred(v_i)}FT_j^c\} \tag{5}$$

其中，RT_i^c 表示任务 v_i 在云端开始执行的最早时间。若 v_j（任务 v_i 的一个直接前驱任务）被局部调度，则 $FT_j^c=0$。因此，式(5)中的 $\max_{v_j\in pred(v_i)}FT_j^c$ 表示 v_i 所有外包至云端的直接前驱任务在云端完成的时间。此外，FT_i^{ws} 表示 v_i 通过无线信道完成外包上传至云端的完成时间，因此，有 $RT_j^c\geqslant FT_i^{ws}$。云端可在时间 RT_j^c 上准确开始调度 v_i，以确保任务间的依赖约束需求。

令 RT_i^{wr} 为云端传回任务 v_i 的数据结果的就绪时间，且：

$$RT_i^{wr}=FT_i^c \tag{6}$$

换言之，云端一旦完成任务执行，立即传回数据结果至移动设备。

3.4　能耗及任务完成时间

若任务 v_i 在设备第 k 个内核上执行，任务带来的能耗为：

$$\begin{aligned}E_{i,k}^l&=P_k\cdot T_{i,k}^l\\&=\alpha_k\cdot(f_k)^{\gamma_k}\cdot T_{i,k}^{l,\min}/\alpha_{k,m}\\&=\alpha_k\cdot(\alpha_{k,m}\cdot f_k^{\max})^{\gamma_k}\cdot T_{i,k}^{l,\min}/\alpha_{k,m}\\&=(\alpha_{k,m})^{(\gamma_k-1)}\cdot\alpha_k\cdot(f_k^{\max})^{\gamma_k}\cdot T_{i,k}^{l,\min}\\&=(\alpha_{k,m})^{(\gamma_k-1)}\cdot E_{i,k}^{l,\max}\end{aligned} \tag{7}$$

其中，$E_{i,k}^{l,\max}$ 为在最大运行频率下任务 v_i 在第 k 个内核执行时的能耗，P_k 和 $T_{i,k}^l$ 均取决于第 k 个内核的运行频率。若任务 v_i 外包至云端执行，移动设备上传任务的能耗为：

$$E_i^c=P^s\cdot T_i^s \tag{8}$$

任务 v_i 在云端执行，则移动设备不产生运动能耗。移动设备运行应用任务的总体能耗 E^{total} 为：

$$E^{\text{total}}=\sum_{i=1}^{N}E_i \tag{9}$$

其中，若任务 v_i 在移动设备的第 k 个内核上执行，则 $E_i=E_{i,k}^l$；若被外包至云端，则 $E_i=E_i^c$。

应用任务的完成时间 T^{total} 为：

$$T^{\text{total}}=\max_{v_i\in\text{exit tasks}}\max\{FT_i^l,FT_i^{wr}\} \tag{10}$$

其中，内层 max 给出出口任务 v_i 的完成时间。若 v_i 在局部内核上调度，则完成时间为 FT_i^l；若 v_i 外包至云端调度，则完成时间为 FT_i^{ws}。

MCC 任务调度问题的任务是：1）决定外包上传至云端的应用任务；2）将剩余任务映射至移动设备的异构内核上；3）决定异构内核和无线通信信道上需要进行的任务调度。其目标是在满足以下约束的同时最小化 E^{total}：1）任务顺序依赖需求；2）应用任务完成时间约束 $T^{\text{total}}\leqslant T^{\max}$，其中，$T^{\max}$ 为允许的最大应用完成时间。

4　MC2ETS 算法的设计

MC2ETS 算法包括 3 个步骤：1）为最小化应用完成时间 T^{total} 进行初始调度；2）在满足应用完成时间约束 $T^{\text{total}}\leqslant T^{\max}$ 的同时，为最小化能耗 E^{total} 进行任务迁移；3）通过在移动设备上执行 DVFS 算法来进一步降低能耗。在 3 个步骤中，必须保证任务间的顺序依赖约束和应用完成时间约束。算法流程如图 2 所示。

为了严格满足应用完成时间约束，步骤 1）的目标仅为最小化 T^{total}，然后在步骤 2）和步骤 3）中通过任务迁移和 DVFS 降低能耗。否则，若步骤 1）首先最小化能耗，由于任务顺序依赖需求和局部内核与无线通信信道上的并行约束，应用完

成时间可能无法严格得到保证。

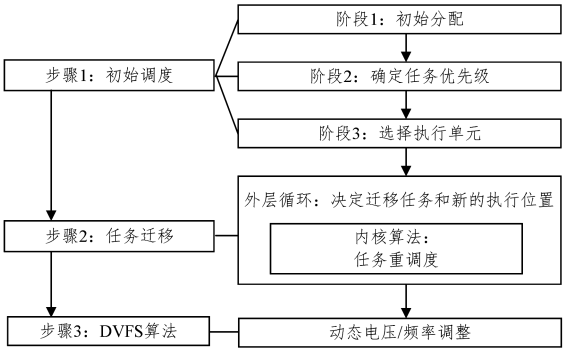

图 2 MC2ETS算法流程

Fig. 2 Flowchart of MC2ETS

4.1 步骤 1:初始调度

初始调度不考虑移动设备中最小延时的调度,具体包括 3 个阶段:初始分配、确定任务优先级和选择执行单元。

1)初始分配

该阶段决定初始分配至云端执行的任务子集,外包该任务集至云端执行可缩短应用完成时间。同时需要注意,该初始分配并非最终的分配方案,这是由于在初始调度步骤的选择执行单元阶段中可以分配更多的任务进行远程云端执行。对于每个任务 v_i,计算其最小局部执行时间 $T_i^{l,\min}$(即在最快的内核上)为:

$$T_i^{l,\min}=\min_{1\leqslant k\leqslant K} T_{i,k}^{l} \tag{11}$$

计算远程云端执行时间 T_i^{re} 为:

$$T_i^{re}=T_i^{s}+T_i^{c}+T_i^{r} \tag{12}$$

若有 $T_i^{re}<T_i^{l,\min}$,则任务 v_i 分配至远程云端执行,并称该任务为"云任务"。

2)确定任务优先级

首先,计算每个任务的计算代价 w_i。若任务 v_i 为云任务,则其计算代价为:

$$w_i=T_i^{re} \tag{13}$$

若任务 v_i 不是云任务,则 w_i 为任务 v_i 在局部内核上的平均计算时间,即:

$$w_i=\underset{1\leqslant k\leqslant K}{avg} T_{i,k}^{l} \tag{14}$$

其中,avg 表示求取平均值。那么,任务 v_i 的优先级递归定义为:

$$priority(v_i)=w_i+\max_{v_j\in succ(v_i)} priority(v_j) \tag{15}$$

其中,$succ(v_i)$为 v_i 的直接后继任务。任务优先级以递归方法从出口任务开始计算。对于出口任务,其优先级为:

$$priority(v_i)=w_i,v_i\in \text{exit tasks} \tag{16}$$

可以理解为:$priority(v_i)$为从任务 v_i 至出口任务间关键路径的长度。

3)选择执行单元

该阶段根据任务优先级的降序排列选择任务并调度。若 v_j 为 v_i 的直接前驱任务,则 $priority(v_j)>priority(v_i)$。因此,当任务 v_i 在该阶段中被选择调度时,其所有直接前驱任务已经被调度。

①若选择的 v_i 为云任务,计算其在无线信道上的就绪时间 RT_i^{ws},分配无线发送信道上的最早可用时槽以上传任务至云端。需要注意的是,由于无线信道可能同时在上传其他任务,移动设备可能无法在时间点 RT_i^{ws} 开始上传任务 v_i。先计算调度的 FT_i^{ws},然后云端将在就绪时间 RT_i^{c} 开始执行 v_i。最后,计算 $FT_i^{c}=RT_i^{c}+T_i^{c}$,且 $FT_i^{wr}=FT_i^{c}+T_i^{r}$。通过这种方式,可以实现调度 v_i 并计算相应的完成时间。

②若选择的 v_i 不是云任务,则可能在内核上调度,也可能在云端调度。此时需要利用情况①中相同的机制估算任务在内核上调度的完成时间和在云端调度的完成时间。然后以完成时间最小化为目标选择将任务调度至局部内核还是云端上执行。同时,在调度时,需要确保满足任务间的顺序依赖约束。

初始调度过程的时间复杂度为 $O(E\times K)$,E 为任务 DAG 中边的数量,K 为移动设备内核数量。对于稀疏 DAG 结构,即 $E=O(N)$,N 为任务数量,初始调度的时间复杂度为 $O(N\times K)$。

下面以图 1 中的任务 DAG 结构为例实现初始调度。假设移动设备拥有 3 个异构内核,表 1 给出了任务在各个内核上的执行时间,即 $T_{i,k}^{l}$的取值,对于所有任务,$T_i^{c}=1$,$T_i^{s}=3$,$T_i^{r}=1$。图 3 给出了任务初始调度结果。例如:任务 v_4 在时间 5 至 12 内在内核 1 上执行,任务 v_2 则被外包上传至云端执行。移动设备利用无线发送信道在时间 5 至 8 内发送 v_2 的计算和输入数量至云端。然后,任务 v_2 在时间 8 至 9 内在云端执行。云端在时间 9 至 10 内将 v_2 的输出数据传回移动设备。该实例中应用的最终完成时间为 18,即为出口任务 v_{10}的完成时间。

表 1 任务在内核上的执行时间

Table 1 Task's execution time on kernels

任务	执行时间/内核 1	执行时间/内核 2	执行时间/内核 3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

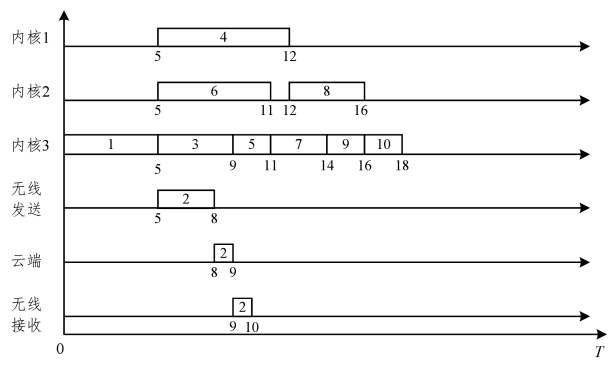

图 3 任务初始调度结果

Fig. 3 Initial task scheduling result

4.2 步骤 2:任务迁移

任务迁移的目标是在满足应用完成时间约束 $T^{\text{total}}\leqslant$

$T^{\max}$的同时最小化能耗 E^{total}。能耗的降低可以通过将任务从一个局部内核迁移至另一内核或云端的方式实现。任务迁移过程是由一个内核算法和外层循环构成的迭代算法。在每次迭代中，外层循环决定迁移的目标任务和新的执行位置（即不同的局部内核或云端），从而最小化能耗 E^{total}，同时确保满足应用完成时间约束 $T^{\text{total}} \leqslant T^{\max}$。对于给定的迁移目标任务和新的执行位置，内核算法的目标是产生新的调度结果，并以线性的时间复杂度最小化应用完成时间 T^{total}。

1）外层循环

任务迁移的外层循环决定从一个内核迁移至其他内核或云端的目标任务，以降低移动设备能耗，同时确保满足应用完成时间约束 $T^{\text{total}} \leqslant T^{\max}$。需要注意的是，任务迁移并未考虑从外包云端的任务迁回内核执行的情况，这是由于此时移动设备的能耗将会增加。

在外层循环的每次迭代中，令 N'为当前调度内核上的任务数量，这些任务可能被迁移至其他 $K-1$ 个内核或云端上执行，因此，总共有 $N' \times K$ 种迁移选择。具体过程如下：

①对于每种选择，运行内核算法寻找新的调度方案，并计算相应的能耗 E^{total}和应用完成时间 T^{total}；

②挑选出当前调度中能耗降低最多并且调度完成 T^{total}无延时现象的迁移选择；

③若无法找到该迁移选择，则选择能耗降低与应用完成时间延长的比率最大的迁移选择；

④在不违背应用完成时间约束的前提下，重复以上步骤直到移动设备的能耗无法进一步最小化为止。

2）内核算法（重调度算法）

令 k_i 为任务 v_i 的执行位置，$k_i \neq 0$ 表明任务 v_i 在第 k_i 个内核上执行，$k_i = 0$ 表明 v_i 外包至云端执行。内核算法拥有一个任务 DAG 的初始调度。给定外层循环中的一个迁移任务 v_{tar}和它的新的执行位置 k_{tar}。内核算法需要产生任务 DAG G 的一个新的调度，其中任务 v_{tar}在新的位置 k_{tar}上执行，剩余任务在初始调度中相同的位置上执行。内核算法的目标是最小化应用完成时间 T^{total}。另一方面，能耗 E^{total}是固定的，已知任务的执行位置后即可通过式(7)－式(9)直接计算能耗。具体过程是：

对于初始调度，利用顺序集 $S_k = \{v(k,1), v(k,2), \cdots\}$表示任务在第 k 个内核上执行的顺序，利用顺序集 $S_0 = \{v(0,1), v(0,2), \cdots\}$表示通过无线发送信道任务外包至云端的顺序。例如，若利用图 3 的调度结果作为初始调度，则有 $S_1 = \{v_4\}$，$S_2 = \{v_6, v_8\}$，$S_3 = \{v_1, v_3, v_5, v_7, v_9, v_{10}\}$，$S_0 = \{v_2\}$。假设任务 v_{tar}在初始调度中在第 k_{ori} 个内核上执行，从外层循环可知 v_{tar}将在新的调度中迁移至第 k_{tar} 个内核上执行。此时应推导新的顺序集 $S_k^{\text{new}}(0 \leqslant k \leqslant K)$，其相当于在新的调度中任务在每个内核执行和无线发送信道上的顺序。在重调度算法中，不改变任务在其他内核上的任务顺序，除了第 k_{tar}个内核上的顺序（v_{tar}将在此内核上执行），即：

$$S_k^{\text{new}} = S_k \backslash v_{\text{tar}}, k = k_{\text{ori}} \tag{17}$$

且

$$S_k^{\text{new}} = S_k, k \neq k_{\text{tar}} \wedge k \neq k_{\text{ori}} \tag{18}$$

接下来，可以通过将 v_{tar}插入到初始调度顺序 $S_{k_{\text{tar}}}$ 中的合适位置上来推导 $S_{k_{\text{tar}}}^{\text{new}}$。同时，需要满足在 k_{tar}内核上的任务顺序依赖约束需求（$k_{\text{tar}} = 0$ 表明无线发送信道）：对于在相同内核上执行或无线通信信道上（传送）的两个任意任务 v_i 和 v_j，若 v_i 是任务 DAG G 中 v_j 的一个中继前驱，则 v_i 必须先于 v_j 进行执行（或传送）。

因此，需要将 v_{tar}插入到 $S_{k_{\text{tar}}}$ 中，使得 v_{tar}在其所有中继前驱之后和其所有中继后继之前执行（或传送）。为了实现该目标，需要计算初始调度中 v_{tar}的就绪时间 RT_{tar}。当 $k_{\text{tar}} > 0$ 时，$RT_{\text{tar}} = RT_{\text{tar}}^{l}$（见式(3)）；当 $k_{\text{tar}} = 0$ 时，$RT_{\text{tar}} = RT_{\text{tar}}^{ws}$（见式(4)）。另外，需要知道在初始调度中每个任务 v_i 的开始时间 ST_i。因此，可推导 $S_{k_{\text{tar}}}^{\text{new}}$为：

$$S_{k_{\text{tar}}}^{\text{new}} = \{v_{(k_{\text{tar}},1)}, \cdots, v_{(k_{\text{tar}},m)}, v_{tar}, v_{(k_{\text{tar}},m+1)}, \cdots\} \tag{19}$$

其中，任务 $v_{(k_{\text{tar}},1)}, \cdots, v_{(k_{\text{tar}},m)}$ 的开始时间早于 RT_{tar}，而任务 $v_{(k_{\text{tar}},m+1)}, \cdots$的开始时间晚于 RT_{tar}。通过这种方式，可以确保第 k_{tar}个内核上的任务顺序依赖约束。

现在结合新的顺序集 $S_k^{\text{new}}(0 \leqslant k \leqslant K)$，可以进一步以线性时间复杂度 O($N$)寻找任务 DAG 的新的调度方案。定义两个矢量 $ready1$ 和 $ready2$。$ready1_i$ 为还未被调度的任务 v_i 的直接前驱的数量。若在相同顺序集 S_k^{new} 中，所有先于 v_i 的任务均已经被调度，则 $ready2_i = 0$。另外，定义一个 LIFO 栈，用于存储已就绪的待调度任务。通过压入满足 $ready1_i = 0$ 和 $ready2_i = 0$ 的任务 v_i 至空栈进行栈的初始化工作，然后重复以下步骤直到栈再次为空，即可调度所有任务：

①从栈中取出任务 v_i。

②假设任务 $v_i \in S_k^{\text{new}}$，若 $k = 0$，调度该任务至无线发送信道，计算移动设备从云端完全接收 v_i 的输出数据的时间；否则，调度该任务至第 k 个内核上执行。

③更新矢量 $ready1$（对于所有 $v_j \in succ(v_i)$的 $ready1_j$ 进行减 1）和 $ready2$，压入所有满足 $ready1_j = 0$ 和 $ready2_j = 0$ 的新任务 v_j 至栈中。

4.3　实例说明

图 4 是 MC2ETS 算法中任务迁移步骤针对图 1 所示的任务 DAG 得到的调度结果。应用完成时间约束设置为 $T^{\text{total}} \leqslant 27$。需要注意的是，图 3 仅代表 MC2ETS 算法的步骤 1）（即初始调度过程）得到的结果，而图 4 是 MC2ETS 算法得到的最终结果。

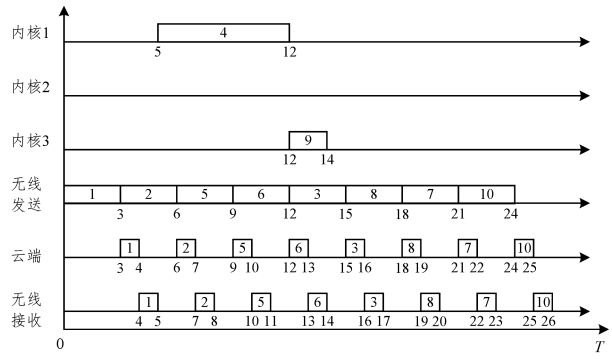

图 4　任务迁移步骤的调度结果

Fig. 4　Scheduling result of migrating tasks

比较图 3 和图 4 可知，为了降低能耗，图 4 中更多的任务被上传至云端执行。图 4 的应用完成时间为 26，高于图 3。

这主要是由于无线发送信道传输速率的限制。内核 1－内核 3 的功耗设置为 $P_1=1, P_2=2, P_3=4$。RF 组件的功耗设置为 $P^s=0.5$。最终，图 3 中 $E^{\text{total}}=100.5, T^{\text{total}}=18$，而图 4 中 $E^{\text{total}}=27, T^{\text{total}}=26$。该结果表明 MC2ETS 算法中的任务迁移过程可以极大地降低能耗，同时满足应用完成时间约束。

4.4 步骤 3：DVFS 算法

初始调度产生了最小化应用完成时间的调度，任务迁移则在局部内核与云端之间进行，降低了应用执行的能耗。MC2ETS 算法的这两个步骤均是假设每个局部内核以最大的运行频率执行任务，利用内核上的 DVFS 能力可以使移动设备在满足应用完成时间约束的同时进一步降低能耗。

若利用 DVFS 降低了高性能内核的执行频率，使得高性能内核拥有与低性能内核相同的性能，则此时高性能内核的能耗仍然高于低性能内核。因此，在任务迁移步骤之后，应该在确定了每个任务的执行单元的前提下应用 DVFS 算法。

任务迁移步骤为了降低能耗而牺牲了应用完成时间，此阶段中产生的调度方案的实际应用完成时间已经非常接近于约束 $T^{\max}$。由前文可知，此时似乎无法进一步降低运行频率从而降低能耗，然而，由于局部内核和无线通信信道上的并行需求，即任务仅能轮流执行或传输，任务 v_i 的实际开始时间 ST_i 可能晚于其就绪时间 RT_i。那么，任务 v_i 的前序任务可能利用 ST_i 与 RT_i 间的间隔以更低的频率执行。

基于步骤 2 产生的调度方案，对于每个局部任务，以升序的方式尝试以每个频率等级执行，并重新调度任务直到找到满足应用完成时间约束的执行频率。若一个任务的执行发生改变，则由于任务顺序依赖，整个调度也可能发生改变。在单个任务的执行频率发生改变后，需要重新调度所有任务。基于此，本文设计了 DVFS 算法以降低该过程的时间复杂度。

基于步骤 2 产生的调度方案，DVFS 算法不改变外包至云端的任务调度和局部任务的开始时间，以避免所有任务发生频繁的重调度。这种调度方案是利用在单个内核上连续执行的两个任务间的时间间隔，其主要思想是对于每个局部任务，如果相同内核上的已完成任务 v_i 与下一任务 v_j 间存在时间间隔，则以升序方式尝试以每个频率等级（共有 M 个频率等级）执行直到找到满足以下条件的执行频率：1）不延长任务 v_j 的开始时间；2）不延长任务 v_i 的后继的开始时间。DVFS 算法的具体过程如算法 1 所示。

算法 1 DVFS 算法

输入：任务迁移的调度结果

输出：局部任务以新的执行频率得到的调度结果

```
1. For 每个局部任务 v_i
2.   flag=0, m=1
3.   While flag==0 & m<M
4.     若 v_i 以第 m 个频率执行，计算新的完成时间 FT_i^new
5.     If 在相同内核上存在新任务 v_j
6.       lim_1 = ST_j
7.     Else
8.       lim_1 = T^max
9.     End if
10.    If v_i 不是出口任务
11.      lim_2 = min_{v_j ∈ succ(v_i)} ST_j
12.    Else
13.      lim_2 = T^max
14.    End If
15.    If FT_i^new ≤ lim_1 & FT_i^new ≤ lim_2
16.      flag=1
17.      分配第 m 个步骤至任务 v_i
18.      更新 v_i 的完成时间
19.    End If
20.  End While
21. End For
```

图 5 是利用 DVFS 算法执行图 1 所示任务 DAG 得到的调度结果。实例中，假设每个内核的运行频率等级 $M=3$，设置频率调整因子为 $\alpha_{k,1}=0.2, \alpha_{k,2}=0.5, \alpha_{k,3}=1, k=1,2,3$，$\gamma_k=2$。比较图 4 和图 5 的结果可知，图 4 中任务 v_9 的执行频率由 $f_k^{\max}$ 变为图 5 中的 $0.5\times f_k^{\max}(k=3)$，其他任务的调度结果并未发生改变。图 4 中 $E^{\text{total}}=27, T^{\text{total}}=26$，而图 5 中 $E^{\text{total}}=23, T^{\text{total}}=26$，这表明 DVFS 算法（MC2ETS 算法的步骤 3）可以进一步降低能耗，同时满足应用完成时间约束。

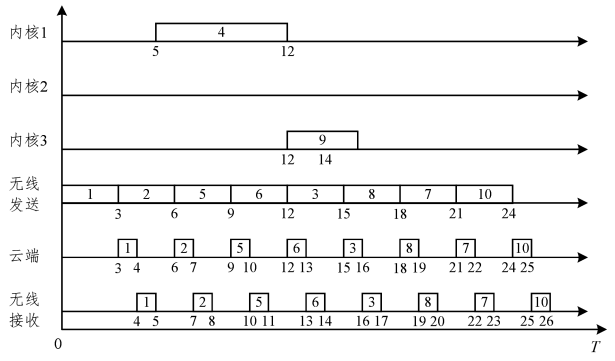

图 5 DVFS 算法调度结果

Fig. 5 Scheduling result of DVFS

5 实验分析

本节通过随机生成的任务 DAG 集验证 MC2ETS 算法的有效性。算法在 MATLAB 中实现，硬件环境配置为 2.6GHz Intel Core-i5 处理器。引入两种基准算法进行性能比较，基准算法 1 的过程描述如下：

1）生成随机矢量 $L, L_i \in \{0,1,\cdots,k,\cdots,K\}$ 表示任务 v_i 的计算位置。若 $L_i=0$，任务 v_i 将外包至云端执行；若 $L_i=k$，则任务 v_i 在移动设备的第 k 个内核上执行。

2）利用修改的初始调度过程在每个局部内核、云端和无线通信信道上顺序调度任务。此处利用的初始调度与 4.1 节中的不同，即每个任务的执行位置预先定义为 L。计算 E^{total} 和 T^{total}。

3）重复步骤 1）－步骤 2）100 00 次，以找到满足约束 $T^{\text{total}} \leqslant T^{\max}$ 的最小化 E^{total}。

通过与基准算法 1 进行比较，可以证明 MC2ETS 算法的有效性和效率。

基准算法 2 除了仅运行于局部移动设备环境中（即移动设备不访问云端，仅利用局部资源进行任务调度）外，与 MC2ETS 算法类似。通过与基准算法 2 进行比较，可以验证 MC2ETS 算法在节省能耗和加强移动设备性能上的优势。

通过随机任务 DAG 发生器得到具有不同特征的任务

DAG 作为测试的任务结构。任务 DAG 发生器的输入参数包括：

1）DAG 中的任务数量 N；

2）DAG 中边的密度 α；

3）移动设备中的内核数量 K；

4）在局部内核上的平均任务执行时间 T_l^{avg}；

5）平均任务发送时间 T_s^{avg}；

6）平均任务接收时间 T_r^{avg}；

7）在云端上的平均任务执行时间 T_c^{avg}。

$T_{i,k}^l$ 产生方式如下：1）对于 $1\leqslant i\leqslant N$ 的 $T_{i,k}^l$ 值以均值 T_l^{avg} 产生；2）将第 $k+1$ 个内核上的 T_{k+1}^l 值设置于 $T_{i,k}^l/\beta$ 周围，$1\leqslant k\leqslant K-1$，$\beta$ 为因子。对于 $1\leqslant i\leqslant N$ 的 $T_i^s/T_i^c/T_i^r$ 值以均值 $T_s^{\text{avg}}/T_c^{\text{avg}}/T_r^{\text{avg}}$ 产生。

假设移动设备的内核数量 $K=3$，内核 1 为低功率内核，内核 3 为高功率内核。3 个内核功耗 P_k 设置为 $P_1=1$，$P_2=2$，$P_3=4$。RF 组件的功耗设置为 $P^s=0.5$。生成拥有不同任务数量 N 和不同特征的 10 个任务 DAG 进行测试。图 6 和图 7 是 3 种算法得到的调度方案对应的应用完成时间 T^{total} 和能耗 E^{total}，同时，图 8 比较了基准算法 1 和 MC2ETS 算法的实际应用执行时间。这里未与基准算法 2 进行比较，因为该算法除了仅是设计在移动设备上执行（不存在云端访问）之外，其他机制与 MC2ETS 算法相同。

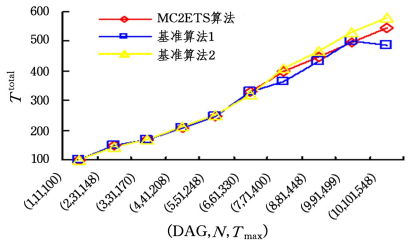

图 6　$K=3$ 时的应用完成时间

Fig. 6　Completion time when $K=3$

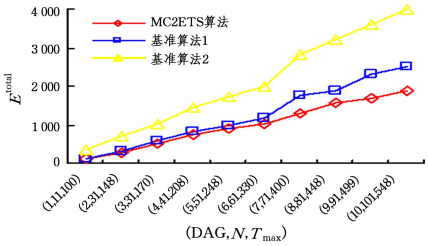

图 7　$K=3$ 时的执行能耗

Fig. 7　Execution energy when $K=3$

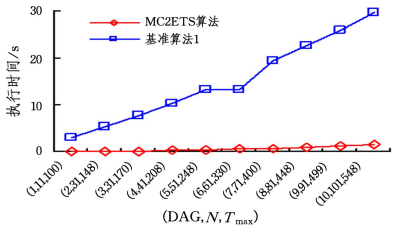

图 8　$K=3$ 时算法的执行时间

Fig. 8　Execution time of algorithm when $K=3$

可以看出，MC2ETS 算法和基准算法 1 可以确保应用完成时间约束。在任务 DAG 2－9 上，MC2ETS 算法拥有比基准算法 1 更低的能耗。然而，对于 DAG 1，MC2ETS 算法的能耗略高，这是由于基准算法 1 在较小的 N 值（$N=11$）下可用穷尽的方式搜索最优解，但这种搜索方式在较大的 N 值时是不可行的。同时，基准算法 1 的任务实际执行时间远高于 MC2ETS 算法。另外，基准算法 2 的调度结果在某些情况下无法满足应用完成时间约束，这表明 MC2ETS 算法可以改进移动设备的性能。同时，与基准算法 2 相比，MC2ETS 算法在能耗降低上最大可以降低约 $\frac{3}{5}$，这也表明 MC2ETS 算法可以极大程度地降低移动设备的能耗。

实验进一步比较了 $K=6$ 时算法的性能。此时，内核 1 是低功率内核，内核 6 是高功率内核。6 个内核的功耗 P_k 的设置为 $P_1=1$，$P_2=2$，$P_3=4$，$P_4=8$，$P_5=16$，$P_6=32$。RF 组件的功耗设置为 $P^s=0.5$。生成拥有不同任务数量 N 和不同特征的 10 个任务 DAG 进行测试。图 9 和图 10 是 3 种算法得到的调度方案对应的应用完成时间 T^{total} 和能耗 E^{total}。同时，图 11 比较了基准算法 1 和 MC2ETS 算法的实际应用执行时间。可以看出，基准算法 1 的某些调度结果（任务 DAG 7 和 DAG 9）无法满足应用完成时间约束，这是由于当内核数量和任务数量相对较大时，随机产生的任务执行位置无法得到较好的结果。基准算法 1 中的一些调度结果将所有任务外包至云端，但也违背了应用完成时间约束。这也是基准算法 1 在任务 DAG 7 和 DAG 9 上的能耗结果低于 MC2ETS 算法的原因。

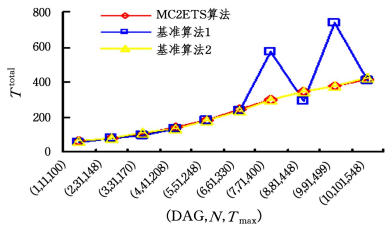

图 9　$K=6$ 时的应用完成时间

Fig. 9　Completion time when $K=6$

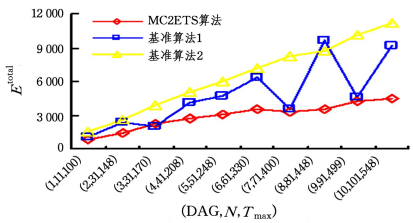

图 10　$K=6$ 时的执行能耗

Fig. 10　Execution energy when $K=6$

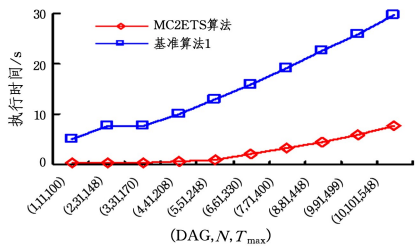

图 11　$K=6$ 时的算法执行时间

Fig. 11　Algorithm execution time when $K=6$

结束语 移动云计算环境中的任务调度不同于单纯传统的任务调度,它可以帮助移动设备上的任务迁移至云端执行。而传统的调度算法缺乏对完成时间和移动设备上调度能耗的同步优化。为了解决这一问题,本文提出了一种移动云计算的能效任务调度算法 MC2ETS。本文通过实例和系统性的实验分析验证了算法的有效性和正确性。进一步的工作可考虑加入更多的约束条件和优化目标,如加入预算约束、优化执行费用,并设计相应的多约束多目标优化调度算法,使调度环境更加贴近于现实的应用环境。

参 考 文 献

[1] PEDRAM M. A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud[C]// Proceedings of the 2013 International Conference on Global Communication. IEEE,2013:2885-2890.

[2] ZARE J,ABOLFAZLI S,SHOJAFAR M,et al. Resource Scheduling in Mobile Cloud Computing:Taxonomy and Open Challenges[C]// IEEE International Conference on Data Science and Data Intensive Systems. IEEE Computer Society,2015:594-603.

[3] BARBERA M V,KOSTA S,MEI A,et al. To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing[J]. Proceedings-IEEE INFOCOM,2015,12(11):1285-1293.

[4] ZHOU B,DASTJERDI A V,CALHEIROS R N,et al. A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service[C]// IEEE,International Conference on Cloud Computing. IEEE,2015:869-876.

[5] ATRE H,RAZDAN K,SAGAR R K. A review of mobile cloud computing[C]// Cloud System and Big Data Engineering. IEEE,2016:199-202.

[6] BALAMURUGAN M,AKILA V. Effective processor selection on heterogeneous computing[C]// International Conference on Science Technology Engineering and Management. IEEE,2016:13-16.

[7] FENG B,GAO J. Distributed Parallel Needleman-Wunsch Algorithm on Heterogeneous Cluster System[C]// International Conference on Network and Information Systems for Computers. IEEE,2016:358-361.

[8] RA M R,SHETH A,MUMMERT L,et al. Odessa:enabling interactive perception applications on mobile devices[C]// International Conference on Mobile Systems, Applications, and Services. ACM,2011:43-56.

[9] YANG L,CAO J,TANG S,et al. A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing[J]. Acm Sigmetrics Performance Evaluation Review,2013,40(4):23-32.

[10] TERZOPOULOS G,KARATZA H D. Dynamic Voltage Scaling Scheduling on Power-Aware Clusters under Power Constraints [C]// IEEE/ACM, International Symposium on Distributed Simulation and Real Time Applications. IEEE,2013:72-78.

[11] GOUDARZI M,ZAMANI M,HAGHIGHAT A T. A fast hybrid multi-site computation offloading for mobile cloud computing[J]. Journal of Network & Computer Applications,2017,80(1):219-231.

[12] SARAVANAN S,VENKATACHALAM V. Advance Map Reduce Task Scheduling algorithm using mobile cloud multimedia services architecture[C]// Sixth International Conference on Advanced Computing. IEEE,2015:21-25.

[13] ELGAZZAR K,MARTIN P,HASSANEIN H S. Cloud-Assisted Computation Offloading to Support Mobile Services[J]. IEEE Transactions on Cloud Computing,2016,4(3):279-292.

[14] LIU Z,ZENG X,HUANG W,et al. Framework for Context - Aware Computation Offloading in Mobile Cloud Computing[C]// International Symposium on Parallel and Distributed Computing. IEEE,2017:172-177.

[15] DESHMUKH N V,DEORANKAR A V. Minimizing energy consumption in transmission efficient wireless sensor network [C]// International Conference on Advances in Electrical,Electronics, Information, Communication and Bio-Informatics. IEEE,2016:475-479.

[16] LI J,LI X,ZHANG R. Energy-and-Time-Saving Task Scheduling Based on Improved Genetic Algorithm in Mobile Cloud Computing[C]// International Conference on Collaborative Computing:Networking, Applications and Worksharing. Springer,Cham,2016:418-428.

[17] KLIAZOVICH D,PECERO J E,TCHERNYKH A,et al. CADAG:Communication-Aware Directed Acyclic Graphs for Modeling Cloud Computing Applications[C]// IEEE Sixth International Conference on Cloud Computing. IEEE,2013:277-284.