说 明 书 摘 要

本发明公开了一种车载云环境下低服务成本的任务调度方法，该方法综合分析车载云环境下任务传输和处理的完整过程，提出了面向完成时间约束的初分配策略，首先使用基于 max-min 和 level 划分的调度方法，通过将 DAG 划分 level、计算车辆间的传输速率和车辆可用性检验这三个方法，完成对每个 level 内任务的最大和最小完成时间的计算。该初分配策略再使用缩短完成时间的调度算法，通过将截止时间根据 level 划分为子截止时间，依据任务调整成功的三个必要条件，将任务完成时间不断缩小，直到满足截止时间要求。本发明还提出了面向低服务成本的重调度算法，通过考虑 level 内重调度前后的排队情况，和跨 level 任务产生的排队现象，将任务重调度以得到较低的服务成本。

说 明 书

一种车载云环境下面向低服务成本的任务调度方法

技术领域

本发明涉及云计算和无线网络领域，特别是涉及一种车载云环境下的任务调度方法。

背景技术

随着计算机技术和通信技术的不断发展进步，移动终端在人们日常生活中正扮演越来越重要的角色。手机、平板电脑、手表等移动终端提供各式各样的服务，如娱乐、生活、办公等，既提供了便利，又实现了其他工具无法完成的功能。在另一方面，日益丰富的移动应用，对移动终端的计算能力、存储容量、电池的电量等带来了巨大挑战。相对于大型电脑，移动设备存在着处理任务能力弱、内存不足、续航能力差等诸多不足。当需要处理计算密集型或数据密集型请求时，移动设备会产生运算效果差、延迟大、宕机等问题，严重影响用户的使用体验。随着车辆数量的不断增加和功能的逐渐增强，可以将这些处理任务和传输数据能力强大、存储空间充足的车辆组合起来，形成车载云来处理用户的请求。车载云计算是一种新型的计算模式，结合了车载网络和移动云计算。车载云计算的出现解决了移动设备能力受限的问题，通过将移动设备无法处理的大型应用，转移到计算能力强大的车载云上执行，在减轻移动设备负担的同时，提升用户体验。

近年来，车载云计算在学术界引起了广泛的关注和讨论。使用车载云计算完成用户的请求，不仅可以提高大型任务的处理能力、有效减少应用完成时间，还能帮助移动设备专注于轻量级任务，提高用户使用的满意度。但是由于车辆的行驶速度很快，车辆作为车载云的资源节点具有不稳定性，经常会出现车载网络不连通的问题。并且由于车载云中车辆数量的局限性，车载云计算仅能提供十分有限的服务。因此，设计一个高效的任务调度算法，有利于充分地利用车载云的计算资源，同时能够有效地提高服务成功率和降低服务成本。

发明内容

有鉴于此，本发明的目的在于提供一种车载云环境下面向低服务成本的任务调度方法，本发明通过面向完成时间约束的初分配策略和面向低服务成本的重调度算法，有效地提高服务成功率并且降低服务成本。

目前车载云环境下的任务调度已经得到了广泛应用，然而现有研究没有综合考虑任务传输和处理的完整过程，或者没有以服务成本作为优化目标展开研究。

不同于现有技术，本发明综合考虑任务的传输和处理，并在满足截止时间的同时，以降低服务成本作为优化目标。本发明在考虑车载云环境的移动性和局限性的同时，考虑到任务的复杂依赖关系，提出一种车载云环境下面向低服务成本的任务调度方法，进行高效的任务调度。

基于上述发明的目的，本发明提出的方法包括如下步骤：

步骤 S1、构建任务模型和车载云系统模型

步骤 S2、给出问题定义

步骤 S3、提出面向完成时间约束的初分配策略

步骤 S4、提出面向低服务成本的重调度算法

进一步地，所述步骤 S1 具体为：

用户需要通过车载云计算处理的应用，可以表示为具有优先级约束的任务工作流，用有向无环图（Directed Acyclic Graph，DAG）表示，并且有 G=(T,E)。其中 $T=\{t_1,t_2,...,t_k\}$表示任务，其中 K 表示任务数。$E\subseteq T\times T$ 表示任务间的依赖关系，如果存在边$e_{i,j}=(t_i,t_j)\in E$，任务t_j需要在任务t_i执行完成后，才能开始执行，那么称任务t_i是任务t_j的前驱任务，任务t_j是任务t_i的后继任务。任务t_i的前驱任务可以表示为 $Pred(t_i)=\{t_p|\exists e_{p,i}\in E\}$，任务$t_i$的后继任务可以表示为 $Succ(t_i)=\{t_s|\exists e_{i,s}\in E\}$。只有当任务的所有前驱任务都执行完成后，任务才能开始执行。如果任务没有前驱任务，则称为入口任务t_{entry}，即 $Pred(t_{entry})=\varnothing$。如果任务没有后继任务，则称为出口任务$t_{exit}$，即 $Succ(t_{exit})=\varnothing$。为了确保一般性，本文假设一个 DAG 只有一个进入任务和一个出口任务。任务t_i的计算负载表示为$L_p(t_i)$，传输负载表示为$L_c(t_i)$。任务依赖矩阵

说 明 书

$I=(I_{i,j})_{K*K}$表示任务间的依赖关系，是 DAG 的邻接矩阵，其中$I_{i,j}>0$表示任务t_i是任务t_j的前驱，且$I_{i,j}$的值为任务t_i的传输负载；$I_{i,j}=0$则表示任务t_i与任务t_j无依赖关系。

许多车辆在一条路段上行驶，可以将这些车辆的运动视为在同一个平面上。车辆作为具有计算能力和传输能力的资源节点，用于处理用户提出的请求，以减轻移动终端的压力。$V=\{v_1,v_2,...,v_m\}$表示车辆，其中 M 表示车辆数。由于车载芯片的 CPU 计算能力和车载存储器的存储能力等的不同，车辆拥有不同的任务处理能力。根据车辆不同的任务处理能力，基于任务处理时间，可以计算出对应的节点服务成本。车辆v_i的处理能力为 $Cap(v_i)$，单位时间的服务成本（unit price）为 $UP(v_i)$。不同车辆节点的单位服务成本不同。在本文中一个任务只能由一个节点处理，不考虑通过节点间合作完成任务。一个节点完成任务处理后，通过无线网络将任务传输到下一个执行任务的节点。车辆v_i与车辆v_j间的传输速率为 $Rate(v_{i,j})$，由两车的物理位置信息和车载节点的网络拓扑决定。路侧单元（RSU）被设置在道路一侧，作为用户、基站与车辆节点的通信中枢。RSU 负责与车载节点进行数据交互，将用户请求的应用传递到车载云中进行处理。考虑到车辆的移动性，车辆的位置信息、车辆节点的网络拓扑信息等由 RSU 负责收集与保存。RSU 通信范围内的车辆构成车载云，一旦车辆离开 RSU 范围，它将无法完成用户任务的传输和处理。Interval 表示车辆在 RSU 范围运动的有效时间间隔。

步骤 S2 具体为：

在车载云处理用户请求的数据传输和任务处理环节中，把任务t_i在车载节点上进行处理的处理时间记作$T_p(t_i)$，任务t_i在两个车载节点间传输的传输时间记作$T_c(t_i)$。

任务的处理时间为任务的计算负载L_p与车辆节点的处理能力 Cap 的比值。任务t_i的处理时间$T_p(t_i)$表达式为：

$$T_p(t_i)=\frac{L_p(t_i)}{Cap(v_i)} \tag{1}$$

类似地，任务的传输时间为任务的传输负载L_c与两车间的传输速率 Rate 的比值。任务t_i的传输时间$T_c(t_i)$表达式为：

$$T_c(t_i)=\frac{L_c(t_i)}{Rate(v_{i,j})} \tag{2}$$

考虑到用户请求在前往车载云环境进行处理之前，首先需要通过与 RSU 进行交互来传递，用户请求在 RSU 与车载云之间的交互时间表示为 InputRSU。InputRSU 可以视为入口任务t_{entry}在处理前所需的一个传输时间，并且 InputRSU 的大小与道路情况有关。

任务必须在其所有的前驱任务都完成数据传输后，才能开始处理。所有的前驱任务都完成数据传输的时刻称为任务的最早开始时间（EST）。任务t_i的最早开始时间 $EST(t_i)$表达式为：

$$EST(t_i)=\begin{cases} InputRSU & ,if\ t_i = t_{entry} \\ \max_{t_p\in Pred(t_i)}\{EST(t_p)+T_p(t_p)+T_c(t_p)\} & ,otherwise \end{cases} \tag{3}$$

任务经过数据传输，传输到任务调度方案中分配的车辆节点后，经过处理达到任务的完成状态，称为任务完成时间（FT）。任务t_i的完成时间 $FT(t_i)$表达式为：

$$FT(t_i)=EST(t_i)+T_p(t_i) \tag{4}$$

应用中的出口任务t_{exit}为所有任务最后一个执行的任务，也是所有任务中任务完成时间最大的一个任务。当出口任务t_{exit}处理完毕，则认为整个应用的处理完毕，t_{exit}的完成时间即为应用完成时间（TFT）。TFT 表达式为：

$$TFT=FT(t_{exit}) \tag{5}$$

车载云服务提供商提供车载云计算服务，并耗费一定的服务成本。每个任务执行消耗的服务成本，和任务在分配到的车辆节点上进行处理的处理时间T_p，以及车辆节点单位时间的服务成本 UP 有关。在车辆v_i上处理任务t_i产生的服务成本 $Cost(t_i)$表达式为：

$$Cost(t_i)=T_p(t_i)*UP(v_i) \tag{6}$$

累加每个任务产生的服务成本，就可以得到处理 DAG 中所有任务产生的总服务成本（TC）。TC 表达式为：

$$TC=\sum_{i=1}^{K} Cost(t_i) \tag{7}$$

当一组具有相互依赖关系的任务，向车载云提出请求时，需要确定任务和车辆节点之间的映射关系，以及节点上的任务执行序列，来使总服务成本 TC 尽可能低。由于用户对任务的实时性有一定要求，会对请求车载云的应用给定一个截止时间（D），作为用户能接受的最大应用完成时间，应用只有在截止时间内完成才算成功。

本文研究任务调度问题的目标，是满足用户定义的截止时间，并追求尽可能低的服务成本。根据公式 5 和公式 7 分别计算应用完成时间 TFT 和总服务成本 TC，本文研究的问题表达式为：

$$\begin{aligned} &\text{Minimize TC} \\ &\text{subject to TFT}\le D \end{aligned} \tag{8}$$

步骤 S3 具体为：

为了提高任务的并行程度，通过将 DAG 划分为多个 level，使每个 level 内的任务之间没有依赖关系。分配任务所属 level 时采用自下而上的方法（Bottom-Up Approach，BUA），序号为 i 的 level 记为l_i，任务t_i所在 level 的序号记为 Level(t_i)，根据任务t_i的前驱依赖关系，level 的计算表达式为：

$$Level(t_i)=\begin{cases} 1 & ,\text{if } t_i = t_{entry} \\ \max_{t_p\in Pred(t_i)}\{Level(t_p)\}+1 & ,\text{otherwise} \end{cases} \tag{9}$$

车辆在道路平面上不停地运动，将车辆抽象为虚拟节点，并基于车辆的位置信息和车辆传输的有效半径，形成车载云的网络拓扑。车辆的位置信息和车辆传输的有效半径，可以通过 RSU 获得。两车辆节点间在网络拓扑中经过边的个数称为传输跳数。网络拓扑中不同的节点之间的传输速率不同，并且由车载云的网络拓扑中节点间的传输跳数决定。通过对车辆网络拓扑图使用最短路径算法，计算两节点间的最短路径，即两车间的最小传输跳数。再通过传输跳数与传输速率的映射，得到传输速率。

由于车辆的移动性，而 RSU 位置固定，存在车辆节点驶出车载云范围的情况。此时车辆为不可用状态，无法传输和处理车载云内的任务。任务的最小前驱任务完成时间（MPFT）是其前驱任务完成时间中最小一个，任务t_i的最小前驱任务完成时间 MPFT(t_i)表达式为：

$$MPFT(t_i)=\min_{t_p\in Pred(t_i)}\{FT(t_p)\} \tag{10}$$

达到任务的最早开始时间后，任务就进入就绪状态。由于一个任务只能在一个车辆节点上执行，并且一个车辆节点不能同时执行多个任务，存在多个任务在同一车辆节点上排队等待执行的情况。正在排队的任务需要等待排队序列中的前一任务完成后，才能使用节点进行任务处理，此时任务的开始时间称为实际开始处理的时间（AST）。排队任务t_i在排队序列中的前一任务为t_j，此时t_i的实际开始处理的时间 AST(t_i)表达式为：

$$AST(t_i)=FT(t_j) \tag{11}$$

发生排队等待后，任务的实际开始时间可能比最早开始时间要晚。实际开始时间与最早开始时间之间的时间称为等待时间（WT）。针对出现排队的情况，任务t_i的等待时间 WT(t_i)表达式为：

$$WT(t_i)=AST(t_i)-EST(t_i) \tag{12}$$

此时任务t_i的完成时间 FT(t_i)表达式为：

$$FT(t_i)=AST(t_i)+T_p(t_i) \tag{13}$$

MPFT(t_i)是任务t_i的最小前驱任务完成时间，FT(t_i)是t_i的任务完成时间，这一时间段内车辆v_j必须在车载云范围内，否则v_j不可用。如果t_i和t_k都在v_j上处理，两个任务进行处理的时间段必须是相错开的，即

没有发生争抢v_j的情况，否则v_j不可用。满足车载云范围内有效和多个任务不争抢同个节点，则认为车辆是可用的。

通过使用将 DAG 划分 level，计算车辆间的传输速率和车辆可用性检验这三个方法，可以完成对每个 level 内任务的最大和最小任务完成时间的计算。Max-min 算法中每个任务分配到车辆节点都会计算一个节点的状态值（State），当对一个 level 内任务的进行第一次分配时，会计算每个任务分配到每个节点的状态值，此时状态值即为任务完成时间。当经车辆可用性检验，车辆为不可用时，状态值为 UNAVAILABLE，意味着不可将这个分配作为后续比较的对象。比较计算得到的可用状态值，分别求一个任务分配到多个节点的 min，和一个 level 内多个任务的 min 中的 max。求出的 max 对应的任务到节点的映射，就完成了一次分配。在l_i内第一次分配中，任务t_j车辆节点v_k的状态值 State(v_k)表达式为：

$$State(v_k)=\begin{cases} FT(t_j) & ,\text{if } v_k \text{ is available} \\ UNAVAILABLE & ,\text{otherwise} \end{cases} \tag{14}$$

第一次分配以后的其他分配，与第一次分配的区别点在于：考虑到任务分配到已处理任务的车辆节点上时，需要在此节点上进行排队，等待其他任务处理完毕后，才能开始处理。此时排队任务的实际开始时间 AST，等于前一个任务的完成时间 FT，如公式 11 所示。

基于 max-min 算法和 level 划分的调度方法，能够在满足任务调度问题复杂约束的前提下，以一种较高效的方式得到了一个可行的初始解。但这种调度方法，求 level 内局部最优的任务完成时间，没有考虑 level 间的多样的依赖关系，很难在全局得到满足截止时间的应用完成时间。

缩短完成时间的调度算法，首先将截止时间根据 level 划分为子截止时间 (SD) 的方法。如果每个 level 内任务完成时间都能够在子截止时间之内，则遍历所有 level 时，整体的应用完成很有可能在截止时间以内。为了满足截止时间，需要对任务进行调整。每次调整一个任务，保持其他任务分配不变，计算该调整方案下任务的预计完成时间（EFT）和预计应用完成时间（ETFT）。

在满足 EFT 小于 SD，或者 ETFT 小于 TFT 后，以及车辆可用性检验后，一次任务调整将可以算作成功，任务将要被标记为“已处理”并加入已处理任务队列。但是由于已处理任务队列中的任务可能在本次调整中，改变原先符合要求的时间。因此，在认定调整成功，将预计任务调度结果更新到当前任务调度结果之前，需要对已处理任务队列中的任务进行检验。当已处理任务队列中的任务完成时间不增加，或者增加但仍小于 SD，则认为任务调整成功。

步骤 S4 具体为：

由于面向完成时间约束的初分配策略中，基于 max-min 和 level 划分的调度方法，和缩短完成时间的调度算法这两个算法，仅从时间层面出发而没有考虑服务成本，导致通过初分配策略后，产生服务成本过高的问题。

重调度策略按 level 的顺序对所有任务进行遍历。如果总服务成本能够降低，同时应用完成时间依然满足截止时间，则认为重调度成功，并将重调度结果更新为算法流程中当前的调度方案。降低服务成本是一个相对值，这种做法使重调度过程中当前的总服务成本，始终保持全局最小。由于计算服务成本相对于计算时间容易很多，这种设计能够筛除很多服务成本上不合适的重调度方案，有效降低算法的计算复杂度。

由于计算服务成本，需要先计算任务的时间相关数据，重调度策略首先从一个 level 内的排队情况开始考虑，计算一个 level 内任务的预计完成时间 EFT。在重新分配某个任务t_a时，由于重调度下同一任务分配到前后两个不同的车辆节点，会引起两个节点上排队任务的时间变化。将l_i内一个车辆节点v_j上的排队任务序列记为 Seq(l_i, v_j)，根据 Seq(l_i, v_j)可以确定原分配方案中排队任务的执行顺序。这两个节点的排队情况有所不同。当新分配到的节点上，有多个任务（包含t_a）需要排队时，根据 FCFS 原则，节点先处理最早开始时间较小的任务。这种做法，保证了车辆节点能够在任务最先需要处理时，就能够及时得到处理，提高节点的资源利用率。而当t_a原分配的节点上，还有多个任务（不包含t_a）在排队时，采用将t_a从节点上的任务序列上移除的方式，保持这些已分配任务的排队顺序不变，按此顺序进行时间的重新计算。

重调度对原分配中排队情况的影响，体现在对于任务序列中重调度任务以前的任务不受影响，对于任务序列以后的排队任务，占用重调度任务的分配时间并重新计算实际开始时间和完成时间。可以发现原分

配中的排队任务，在少了一个参与排队的任务后，其他排队任务完成时间应该小于等于它们原来的完成时间。因此，在重分配的情况下，它们的完成时间不作为重分配方案是否有效的判断依据。对任务进行重调度，由于重分配前后任务序列的变化，以及任务开始时间和完成时间的改变，需要重新经过车辆可用性检验，保证更新后这些排队任务分配到的节点是可用的。

通过上述对重调度中排队情况的考虑，可以计算出一个 level 内的任务预计完成时间。发生一次重调度的 level 中，在考虑了排队问题后，这个 level 内的所有任务就有了重调度后的结果。

对于重调度的后续 level，下一 level 的最早开始时间与前 level 的任务完成时间相关，首先要更新下一 level 内所有任务的最早开始时间。并且由于重调度会改变任务分配的节点，跨 level 的任务也存在在同一节点排队的问题。由于本重调度方案是按 level 的顺序进行遍历，位于后续 level 的任务，应当排队在前 level 任务之后。按照这样的规则，在其他任务分配结果不变的情况下，更新任务开始时间和完成时间，并检验车辆的可用性。

这样，计算全部 level 内的任务处理时间和完成时间，就可以得到预计应用完成时间 ETFT 和预计总服务成本 ETC。一个关键点是要求重调度的应用完成时间仍然满足截止时间，然后考虑总服务成本是否降低。满足这两个条件，并且车辆节点通过车辆可用性检验后，则认为一次重调度成功。

本发明的有益效果是：

1、现有技术仅考虑降低系统成本或降低延时，本发明综合考虑在用户可接受的截至时间内最小化服务成本，并考虑车载云环境的移动性和异构性，设计出适合车载云环境的任务调度方法。

2、本发明提出了一种面向车载云环境的任务调度方法，解决了车载云下复杂的任务调度问题。本发明和现有方案相比，本发明具有较低的服务成本并能有效减少任务完成时间。

附图说明

图 1 为本发明的流程图

具体实施方式

为使本发明实施例的目的、技术方案和优点更加清楚，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例是本发明一部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例，都属于本发明保护的范围。

在正式说明实施例之前，先介绍本发明提出的缩短完成时间的调度算法和面向低服务成本的重调度算法。

缩短完成时间的调度算法如 Algorithm 1 所示：

Algorithm 1-缩短完成时间的调度算法

Input: K 个任务:$t_1,t_2,...,t_k$，任务依赖矩阵:I; M 个车载节点:$v_1,v_2,...,v_m$; 各 level 内的任务:tasksInLevels; 任务截止时间:D; 基于 level 的任务子截止时间:SD; 原任务调度结果:P

Output: 缩短完成时间后的任务调度结果:P

1: **if**(TFT>D)

2: 记划分的level数为levelNum

3: **for**(i=1; i<=levelNum; i++)

4: 记p中的任务数为taskNum

5: **for**(j=1; j<=taskNum; j++)

6: 求满足max(FT-SD(l_i))的任务，记该任务为t_m，对t_m进行调整

7: **end for**

8: **for**(j=1; j<=M; j++)

9: 计算EFT(t_m)和ETFT

10: 如果存在一个分配使EFT(t_m)<SD(l_i)，则flag=1

11: **end for**

12: **if**(flag==1)

13: 求满足min(SD(l_i)-EFT(t_m))的分配

14: **else**

15: 求满足min(EFT(t_m)-SD(l_i))的分配

16: 如果此时ETFT>TFT，则此调整方案无效

17: 如果q中已处理任务的完成时间都不增加，或仍满足小于SD(l_i)，则flag=1

18: **if**(flag==1)

19: 任务调整成功，将任务调度结果更新到P中

20: **if**(ETFT<=D)

21: 任务调度结果满足截止时间，退出循环

22: **else**

23: 任务调整失败

24: 将t_m标记为“已处理”并加入q，接着遍历p中的下一个任务

25: **end for**

26:返回P

其中，TFT 是基于 max-min 算法和 level 划分的调度方法得到的应用完成时间，是本调度策略需要减小的目标。D 是截止时间，将 TFT 缩小到 D 以内，算法即完成目标。p 是一个 level 内的未标记任务队列，q 是一个 level 内的已处理任务队列，用于区别任务是否已进行调整。FT 是任务完成时间，SD()是的子截止时间，算法第 8 行求 max(FT-SD())的任务进行调整，为了优先调整最不满足子截止时间的任务。EFT)是的预计完成时间，ETFT 是预计应用完成时间。算法第 15 和第 17 行，求 min 是为了使完成时间改变量最小。算法第 22 行会在每次任务调整成功后，都进行 ETFT 和 D 的比较，及时退出算法剩余循环。

面向低服务成本的重调度算法如 Algorithm 2 所示：

Algorithm 2-面向低服务成本的重调度算法

Input: K 个任务:$t_1,t_2,...,t_k$，任务依赖矩阵:I; M 个车载节点:$v_1,v_2,...,v_m$；各 level 内的任务:tasksInLevels；任务截止时间:D；原任务调度结果:P

Output: 面向低服务成本的重调度结果:P

1: 记划分的level数为levelNum

2: **for**(i=1; i<=levelNum; i++)

3: 记p中的任务数为taskNum

4: **for**(j=1; j<=taskNum; j++)

5: 求负载最大的任务t_m，对t_m进行重调度

6: **end for**

7: **for**(j=1; j<=M; j++)

8: 计算$T_p(t_m)$和$T_c(t_m)$，计算EST(t_m)

9: 计算EFT(t_m)和level内其他任务的EFT

10: 计算ETFT和ETC

11: **if**(ETC<TC&&ETFT<D)

12: 此重分配可行，保留此重调度方案，将v_j加入r中

13: **end for**

14: **if**(r不为空)

15: 求r中使ETC最小的一个节点v

16: t_m->v的重调度方案成功，将对应的任务调度结果更新到P中

17: **else**

18: 重分配失败

19: 将t_m标记为“已处理”并加入q，接着遍历p中的下一个任务

20: **end for**

21:**end for**

22:返回P

其中，TC 是面向完成时间约束的初分配策略的总服务成本，是本调度策略需要降低的目标。p 是一个 level 内的未标记任务队列，q 是一个 level 内的已处理任务队列，用于区别任务是否已进行重调度。r 是一次重调度中可行的节点队列，当有多个节点满足要求时，选择使 ETC 最小的调度方案。对于 DAG 中的所有任务，面向低服务成本的重调度算法，采用按照 level 的顺序进行迭代式的重调度。对于同一个 level，采用负载大小降序的顺序，选择任务进行重调度。大负载任务优先重调度，目标是先尽量取得较低的总服务成本，降低重调度其他任务的计算复杂度（其他任务的重调度先计算总服务成本，如果相对当前总服务成本没有降低，则直接舍弃该方案，无需计算应用完成时间）。$T_p(t_m)$是将任务t_m分配到车辆v_j的处理速率，$T_c(t_m)$是t_m前驱任务分配的车辆与v_j间的传输速率。EST(t_m)是t_m的最早开始时间，EFT(t_m)是t_m的预计完成时间，ETFT 是此重调度方案下预计应用完成时间，ETC 是预计总服务成本。算法第 11 行是重调度成功的判断条件。

实施例 1

本实施例 1 提供了一种车载云环境下面向低服务成本的任务调度方法，包括如下步骤:

步骤 S1、构建任务模型和车载云系统模型；

步骤 S2、给出问题定义；

具体的说，步骤 S2 为:

在车载云处理用户请求的数据传输和任务处理环节中，把任务t_i在车载节点上进行处理的处理时间记作$T_p(t_i)$，任务t_i在两个车载节点间传输的传输时间记作$T_c(t_i)$。

任务的处理时间为任务的计算负载L_p与车辆节点的处理能力 Cap 的比值。任务t_i的处理时间$T_p(t_i)$表达式为:

$$T_p(t_i)=\frac{L_p(t_i)}{Cap(v_i)} \tag{1}$$

类似地，任务的传输时间为任务的传输负载L_c与两车间的传输速率 Rate 的比值。任务t_i的传输时间$T_c(t_i)$表达式为:

$$T_c(t_i)=\frac{L_c(t_i)}{Rate(v_{i,j})} \tag{2}$$

考虑到用户请求在前往车载云环境进行处理之前，首先需要通过与 RSU 进行交互来传递，用户请求在 RSU 与车载云之间的交互时间表示为 InputRSU。InputRSU 可以视为入口任务t_{entry}在处理前所需的一个传输时间，并且 InputRSU 的大小与道路情况有关。

任务必须在其所有的前驱任务都完成数据传输后，才能开始处理。所有的前驱任务都完成数据传输的时刻称为任务的最早开始时间（EST）。任务t_i的最早开始时间 EST(t_i)表达式为：

$$EST(t_i)=\begin{cases} InputRSU & ,if\ t_i = t_{entry} \\ \max_{t_p\in Pred(t_i)}\{EST(t_p)+T_p(t_p)+T_c(t_p)\} & ,otherwise \end{cases} \tag{3}$$

任务经过数据传输，传输到任务调度方案中分配的车辆节点后，经过处理达到任务的完成状态，称为任务完成时间（FT）。任务t_i的完成时间 FT(t_i)表达式为：

$$FT(t_i)=EST(t_i)+T_p(t_i) \tag{4}$$

应用中的出口任务t_{exit}为所有任务最后一个执行的任务，也是所有任务中任务完成时间最大的一个任务。当出口任务t_{exit}处理完毕，则认为整个应用的处理完毕，t_{exit}的完成时间即为应用完成时间（TFT）。TFT 表达式为：

$$TFT=FT(t_{exit}) \tag{5}$$

车载云服务提供商提供车载云计算服务，并耗费一定的服务成本。每个任务执行消耗的服务成本，和任务在分配到的车辆节点上进行处理的处理时间T_p，以及车辆节点单位时间的服务成本 UP 有关。在车辆v_i上处理任务t_i产生的服务成本 Cost(t_i)表达式为：

$$Cost(t_i)=T_p(t_i)*UP(v_i) \tag{6}$$

累加每个任务产生的服务成本，就可以得到处理 DAG 中所有任务产生的总服务成本（TC）。TC 表达式为：

$$TC=\sum_{i=1}^{K} Cost(t_i) \tag{7}$$

当一组具有相互依赖关系的任务，向车载云提出请求时，需要确定任务和车辆节点之间的映射关系，以及节点上的任务执行序列，来使总服务成本 TC 尽可能低。由于用户对任务的实时性有一定要求，会对请求车载云的应用给定一个截止时间（D），作为用户能接受的最大应用完成时间，应用只有在截止时间内完成才算成功。

步骤 S3、提出面向完成时间约束的初分配策略；

具体的说，步骤 S3 为：

为了提高任务的并行程度，通过将 DAG 划分为多个 level，使每个 level 内的任务之间没有依赖关系。分配任务所属 level 时采用自下而上的方法（Bottom-Up Approach，BUA），序号为 i 的 level 记为l_i，任务t_i所在 level 的序号记为 Level(t_i)，根据任务t_i的前驱依赖关系，level 的计算表达式为：

$$Level(t_i)=\begin{cases} 1 & ,if\ t_i = t_{entry} \\ \max_{t_p\in Pred(t_i)}\{Level(t_p)\}+1 & ,otherwise \end{cases} \tag{9}$$

由于车辆的移动性，而 RSU 位置固定，存在车辆节点驶出车载云范围的情况。此时车辆为不可用状态，无法传输和处理车载云内的任务。任务的最小前驱任务完成时间（MPFT）是其前驱任务完成时间中最小一个，任务t_i的最小前驱任务完成时间 MPFT(t_i)表达式为：

$$MPFT(t_i) = \min_{t_p\in Pred(t_i)}\{FT(t_p)\} \tag{10}$$

达到任务的最早开始时间后，任务就进入就绪状态。由于一个任务只能在一个车辆节点上执行，并且一个车辆节点不能同时执行多个任务，存在多个任务在同一车辆节点上排队等待执行的情况。正在排队的

说 明 书

任务需要等待排队序列中的前一任务完成后，才能使用节点进行任务处理，此时任务的开始时间称为实际开始处理的时间（AST）。排队任务t_i在排队序列中的前一任务为t_j，此时t_i的实际开始处理的时间 AST(t_i)表达式为：

$$AST(t_i)=FT(t_j) \tag{11}$$

发生排队等待后，任务的实际开始时间可能比最早开始时间要晚。实际开始时间与最早开始时间之间的时间称为等待时间（WT）。针对出现排队的情况，任务t_i的等待时间 WT(t_i)表达式为：

$$WT(t_i)=AST(t_i)-EST(t_i) \tag{12}$$

此时任务t_i的完成时间 FT(t_i)表达式为：

$$FT(t_i) =AST(t_i)+T_p(t_i) \tag{13}$$

MPFT(t_i)是任务t_i的最小前驱任务完成时间，FT(t_i)是t_i的任务完成时间，这一时间段内车辆v_j必须在车载云范围内，否则v_j不可用。如果t_i和t_k都在v_j上处理，两个任务进行处理的时间段必须是相错开的，即没有发生争抢v_j的情况，否则v_j不可用。满足车载云范围内有效和多个任务不争抢同个节点，则认为车辆是可用的。

Max-min 算法中每个任务分配到车辆节点都会计算一个节点的状态值（State），当对一个 level 内任务的进行第一次分配时，会计算每个任务分配到每个节点的状态值，此时状态值即为任务完成时间。当经车辆可用性检验，车辆为不可用时，状态值为 UNAVAILABLE，意味着不可将这个分配作为后续比较的对象。比较计算得到的可用状态值，分别求一个任务分配到多个节点的 min，和一个 level 内多个任务的 min 中的 max。求出的 max 对应的任务到节点的映射，就完成了一次分配。在l_i内第一次分配中，任务t_j车辆节点v_k的状态值 State(v_k)表达式为：

$$State(v_k)=\begin{cases} FT(t_j) & ,\text{if } v_k \text{ is available} \\ UNAVAILABLE & ,\text{otherwise} \end{cases} \tag{14}$$

缩短完成时间的调度算法，首先将截止时间根据 level 划分为子截止时间（SD）的方法。如果每个 level 内任务完成时间都能够在子截止时间之内，则遍历所有 level 时，整体的应用完成很有可能在截止时间以内。为了满足截止时间，需要对任务进行调整。每次调整一个任务，保持其他任务分配不变，计算该调整方案下任务的预计完成时间（EFT）和预计应用完成时间（ETFT）。

任务调整成功有两个必要条件：第一个条件是调整后 EFT(t_j)<SD(l_i)，或者 EFT(t_j)>=SD(l_i)但 ETFT<TFT；第二个条件是所有分配经车辆可用性检验，车辆均为可用。

不断调整任务，直到满足截止时间或者遍历所有任务。

步骤 S4、提出面向低服务成本的重调度算法；

本发明未详述之处，均为本领域技术人员的公知技术。

以上详细描述了本发明的较佳具体实施例。应当理解，本领域的普通技术人员无需创造性劳动就可以根据本发明的构思作出诸多修改和变化。因此，凡本技术领域中技术人员依本发明的构思在现有技术的基础上通过逻辑分析、推理或者有限的实验可以得到的技术方案，皆应在由权利要求书所确定的保护范围内。