

物联网环境下具有顺序约束关系的静态任务表调度算法

叶佳*, 周鸣争

(安徽工程大学 计算机与信息学院, 安徽 芜湖 241000)

(* 通信作者电子邮箱 yjhgl23@yeah.net)

摘要:针对物联网异构调度环境下并行计算的静态任务调度问题,提出了一种基于最早完成时间策略改变调度顺序的表调度算法 HDPTS。该算法针对现有表调度算法在调度前不能准确地确定调度顺序的问题,在 IHEFT 算法的基础上添加了一个动态优先级调度策略,当节点的前驱任务都已经完成调度任务时,就改变该节点的调度优先级。任务优先级的计算在所有前驱任务到达这个任务的最晚完成时间与所有资源上最大可以使用时间之间取最大值的基础上,同时考虑到分配到各个资源上的任务对后继任务的影响和资源上的负载情况,以及上行权重的计算值和对出口任务的影响,使得优先级计算更加合理,能够根据任务分配动态合理改变任务调度顺序。通过随机生成一个算例进行测试,结果表明 HDPTS 比 IHEFT、HEFT 在调度长度方面减少 14.29%;对大量随机产生的特定结构的有向无环图(DAG)进行测试,测试结果显示 HDPTS 算法比 IHEFT、HEFT 和 LDCP 算法更有效。

关键词:异构环境;物联网;任务调度;表调度;静态调度

中图分类号: TP301.6 **文献标志码:** A

List scheduling algorithm for static task with dependence in Internet of things environment

YE Jia*, ZHOU Mingzheng

(College of Computer and Information Engineering, Anhui Polytechnic University, Wuhu Anhui 241000, China)

Abstract: The static task list scheduling problems in distributed heterogeneous computing environment of Internet of things was studied, and a list scheduling algorithm named Heterogeneous Dynamic Priority Task Scheduling (HDPTS) was proposed, which can dynamically change scheduling sequence based on the strategy of the earliest completion time. Concerning that the existing list scheduling algorithms can not accurately determine the scheduling order before scheduling, on the basis of Improved Heterogeneous Earliest Finish Time (IHEFT) algorithm, a dynamic priority scheduling policy was added to it. When precursor tasks of a node completed scheduling, the scheduling priority of this node should be changed. Scheduling priority of task was calculated on the basis of choosing the maximum value between the latest completion time of all immediate predecessor tasks and the maximum available time of all the resources. At the same time, some other factors were also considered, including the influence to the subsequent tasks of the tasks assigned to the resource, the resource load, the calculated value of uplink weight and the influence to the exit tasks. All these considerations make the priority calculation be more reasonable, so as to dynamically change the task scheduling sequence reasonably according to the task allocation situation. By a randomly generated example test, the results show that the scheduling length of HDPTS reduced by 14.29% compared with IHEFT, HEFT (Heterogeneous Earliest Finish Time); the test results on a large number of randomly generated Directed Acyclic Graph (DAG) with specific structure prove that HDPTS is more effective than IHEFT, HEFT and LDCP (Longest Dynamic Critic Path) algorithms.

Key words: heterogeneous environment; Internet of Things (IoT); task scheduling; list scheduling; static scheduling

0 引言

物联网利用射频识别标签、传感器、制动器、移动电话等设备,把普遍存在我们周围的各种事物或对象串联起来,实现它们之间的交流与合作,构成异构分布式处理环境,进而完成共同的任务。在处理用户应用请求时,将任务分解为若干互相依赖的任务进行并行处理,这样可以提高系统的处理性能,同时也保证任务执行的可靠性,但这种异构计算资源系统为任务调度带来了新的挑战^[1]。

用户请求应用这类任务一般可以分解为若干相互依赖的子任务,可以通过有向无环图(Directed Acyclic Graph, DAG)来描述,其中每个节点表示一个子任务,节点之间的有向边表示子任务之间的依赖关系。一般多机调度问题是 NP 完全问题^[2],物联网环境下的异构调度问题更为复杂。目前任务调度可被分成两大类:静态调度^[3]和动态调度^[4]。根据 DAG,计算每个子任务在各个资源上的处理时间,以及子任务之间的通信时间,采用非抢占式调度,则称为静态调度;否则称为抢占式调度^[5]或者动态调度。静态任务调度相对于动态调

收稿日期:2014-04-11;**修回日期:**2014-06-18。 **基金项目:**国家自然科学基金资助项目(61300170);安徽省自然科学基金资助项目(1308085MF88,1408085MF124);安徽省教育厅自然科学基金资助项目(KJ2013A040)。

作者简介:叶佳(1988-),男,安徽黄山人,硕士研究生,主要研究方向:计算机控制、嵌入式系统;周鸣争(1958-),男,安徽安庆人,教授,主要研究方向:计算机控制、嵌入式系统、计算机网络、信息安全。

度实现更为简单,开销更小^[6]。基本静态调度算法可以分为基于随机搜索和基于启发法两类:基于随机搜索算法包括基因算法、退火算法、局部搜索技术等;基于启发法的算法包括表调度法、聚簇法、任务复制法等。

表调度算法以较低的复杂度和较高的调度效率得到广泛研究,目前已有许多针对异构处理环境下的表调度算法,比如 HEFT (Heterogeneous Earliest Finish-Time)^[7]、CPOP (Critical-Path-on a Processor)^[7]、LDCP (Longest Dynamic Critical Path)^[8]、IHEFT (Improved Heterogeneous Earliest Finish-Time)^[9]等。HEFT 与 CPOP 是异构环境下的经典任务调度算法,经过证实其调度结果优于之前所提出的很多异构环境下的实时任务调度算法;但 HEFT 与 CPOP 算法的调度顺序在调度前就已经确定,后续调度过程中没有优化调度顺序,所以对于复杂的 DAG 调度结果并不理想。LDCP 算法也用来处理异构环境下具有依赖关系的任务调度问题,该算法为每个资源都定义一个 DAGP (Directed Acyclic Graph that corresponds to a Processor),通过关键 DAGP 进行任务排序;但是该算法更适合任务在资源上的处理时间具有单调性的异构环境中,而物联网环境中资源上的处理时间往往具有非单调性。IHEFT 算法修改了 HEFT 算法中节点分配到资源上的优先级,能获得比 HEFT 更好的调度结果;但是同样由于调度顺序是在调度前已经确定了,没有在调度过程中适当修改调度顺序,所以调度结果不是很理想;虽然与 HEFT 相比,IHEFT 算法考虑到了前驱的影响,但该优先级在任务调度过程中并没有发生变化,所以选择方式上并不合理。

除了这些典型的表调度算法外,还有基于智能搜索的调度算法,如文献[10-11]提出的都是基于粒子群优化的任务调度算法;还有基于复制的调度算法,如文献[12]提出的 WPTS (Weighted Priority Task Scheduling) 算法在计算任务选择优先级时,利用加权方式,将任务节点在资源上的平均执行开销、前驱的平均接收开销以及与后继的平均传输开销加权在一起,按优先级大小来选择调度任务节点,采用复制任务到空闲时间片上或者复制前驱的方式,从而提高调度性能;此外面向 DAG 的调度方法还有表调度和复制结合的方法^[13]、贝叶斯优化算法^[14]、免疫克隆选择^[15]、随机调度算法^[16]等。

本文针对现有表调度算法的不足,提出一种针对资源对任务处理时间具有非单调性的异构处理环境下的动态优先级任务调度算法 HDPTS (Heterogeneous Dynamic Priority task Scheduling)。该算法在调度过程中实现动态优化调度顺序,采用最早完成时间优先策略进行调度^[6],通过随机生成的一个算例说明调度过程和效果,最后通过随机生成大量特定结构的 DAG 进行实验对比,结果分析表明 HDPTS 算法比 IHEFT、HEFT 和 LDCP 调度效果更好。

1 问题描述

本文将一组有序任务用一个 DAG 来表示,从而将任务调度问题转为调度 DAG 的问题,即 $G = (T, E)$ 。其中: $T = \{t_i, i = 1, 2, \dots, n\}$ 是有序任务的集合, $n = |T|$ 表示 DAG 中节点的数目; $E = \{e_{i,j}\}$ 是边的集合, $e_{i,j}$ 表示任务 t_i 到 t_j 的边,且任务 t_i 与 t_j 具有依赖关系,即任务 t_j 必须在 t_i 完成后才能开始

执行。

假设物联网中资源之间通过同构高速网络连接,且任意两个资源之间的通信速率相同,那么 DAG 中每条边对应一个通信开销, $C = \{c_{i,j}\}$ 表示整个任务图的通信开销集合,其中 $c_{i,j}$ 描述任务 t_i 到任务 t_j 所需要的通信开销,任务 t_i 为任务 t_j 的前驱任务。设物联网所拥有的异构资源集合为 $R = \{r_j, j = 1, 2, 3, \dots, m\}$, 其中: r_j 是资源, $m = |R|$ 表示资源数目。各个资源之间的处理能力不一样,且存在偏好,用 $w_{i,j}$ 表示任务 t_i 在资源 r_j 上的处理时间,那么 $W = [w_{i,j}]_{n \times m}$ 是任务集合在资源上的处理时间矩阵,如图 1 和表 1 所示。

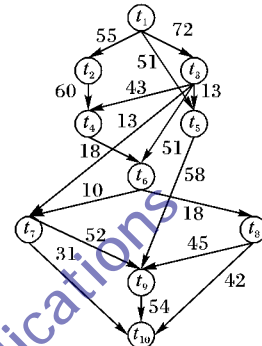


图1 一个任务 DAG 示意图

表1 图1中任务在不同资源上的处理时间

任务	r_1	r_2	r_3	任务	r_1	r_2	r_3
t_1	6	13	22	t_6	20	44	18
t_2	44	42	15	t_7	45	38	48
t_3	21	41	40	t_8	23	33	48
t_4	8	37	40	t_9	45	8	36
t_5	14	26	48	t_{10}	14	5	45

本文中 t_i 的前驱和后继任务分别描述为:

$$pred(t_i) = \{t_j | t_j \in T, e_{ji} \in E\}$$

$$succ(t_i) = \{t_j | t_j \in T, e_{ij} \in E\}$$

入口任务描述为 t_{entry} , 出口任务描述为 t_{exit} , 那么 $pred(t_{entry}) = \emptyset, succ(t_{exit}) = \emptyset$ 。

DAG 上节点任务 t_i 在资源上的最早开始时间描述为:

$$\begin{cases} EST(t_i, r_j) = \min\{t | t \geq \max_{t_m \in pred(t_i)} (AFT(t_m) + c_{m,i}'), \\ \quad idel^{(r_j)}(t, t + w_{i,j})\} \\ EST(t_{start}, r_j) = 0, r_j \in R \end{cases} \quad (1)$$

其中 $AFT(t_m)$ 表示任务 t_m 的实际完成时间。当 t_m 实际调度资源与 t_i 一致,即为 r_j 时, $c_{m,i}' = 0$; 否则 $c_{m,i}' = c_{m,i}$, 表示资源 $idel^{(r_j)}(t, t + w_{i,j})$ 在时间段 $[t, t + w_{i,j}]$ 中处于空闲状态。最早完成时间为

$$EFT(t_i, r_j) = EST(t_i, r_j) + w_{i,j} \quad (2)$$

2 HDPTS 实现基本原理

HDPTS 算法的基本思想如下:第一阶段是任务上行权值的计算阶段,主要包括任务分层、任务上行权值的计算;第二阶段是任务调度阶段,主要是构造调度顺序表,将任务分配到资源上,并不断优化后续任务,即在分配过程中动态改变任务分配的优先级顺序,是对第一阶段上行权重计算的优化,使分配先后顺序更为合理,保证得到更好的调度结果。

2.1 任务上行权重计算阶段

2.1.1 任务节点分层

任务分层是为方便后续任务分配过程中动态计算任务分配的上行权重,任务分层的计算公式如下:

$$level(n_i) = \begin{cases} 1, & n_i = n_s \\ \max_{n_j \in pred(n_i)} \{level(n_j)\} + 1, & \text{其他} \end{cases} \quad (3)$$

2.1.2 任务权重计算

本文在分配前给每个任务节点分配一个自下而上的任务上行权重,便于建立一个初步的调度表,也便于后续分配过程中动态优化调度优先级,分配前的上行权重计算方法如下:

$$\begin{cases} rank_{up}(t_i) = \min_{r_j \in R} \max_{t_k \in succ(t_i)} \{rank_{up}(t_k) + w_{i,j} + c_{i,k}'\} \\ rank_{up}(t_{exit}) = \min_{r_j \in R} w_{exit,j} \end{cases} \quad (4)$$

其中:

$$c_{i,k}' = \begin{cases} 0, & r_j = pred_R(t_k) \\ c_{i,k}, & \text{其他} \end{cases}$$

$pred_R(t_k)$ 表示使得上行权重最小的资源,称为预分配资源。倘若用 $rank_{up}(t_i, r_j) = \max_{t_k \in succ(t_i)} \{rank_{up}(t_k) + w_{i,j} + c_{i,k}'\}$ 表示 t_i 分配给 r_j 的上行权重,则 $pred_R(t_k)$ 对于 $\forall r \in R, rank_{up}(t_i, r) \leq rank_{up}(t_i, pred_R(t_k))$ 成立。

2.2 任务调度与调度优先级优化

2.2.1 构造调度顺序表

HDPTS 算法首先用栈建立任务调度顺序表 pStack,然后逐层将任务进行入栈;入栈完毕后,依次出栈计算节点初始上行权重值,然后根据节点上行权重值从小到大依次入栈;然后出栈,边调度边优化调度顺序,实现动态优化顺序,使调度结果更好。

算法根据式(4)计算每个节点的上行权重值,权重值越高的调度优先级越高,即按照权重的降序排列确认任务调度的优先级序列。

计算每个节点的上行权重值时,HDPTS 算法采用栈这种数据结构建立节点的初始化优先级序列。首先建立一个空栈 pStack,从整个任务系统按层次从开始节点开始,即第一层节点开始入栈,然后第二层,直到最后一层入栈完毕;然后依次出栈计算上行权重值,这样保证每个节点计算上行权重时,它的后继节点都已经计算完毕。等栈空后,则上行权重值计算完毕,对任务节点按上行权重值升序的方式进行排序,如果权重值相同,则后继节点较多的后入栈。这样能保证上行权重值较高的较先执行的原则,即将上行权重值作为初始优先级值。该计算上行权重值过程如以下伪码所示:

```
for each task  $t_i$  in the pStack
   $rank_{up}(t_i) = \max Value$ ; //每个任务的权重值初始为无穷大
  for each resource  $r_j \in R$ 
     $rank_{up} = 0$ ; //任务  $t_i$  在资源  $r_j$  上的权重值初始化为 0
    if  $t_i = t_{exit}$  //  $t_{exit}$  是整个 DAG 的出口任务
       $rank_{up} = w(i, j)$ ;
    else
      for each task  $t_k \in succ(t_i)$ 
        if  $pred\_R(t_k) = r_j$  //后继任务  $t_k$  在资源  $r_j$  上时,  $rank_{up1}$ 
          //为对应前驱  $t_k$  计算得到的权重值
           $rank_{up1} = rank_{up}(t_k) + w_{i,j}$ ;
      //根据式(4)计算与后继任务处于同一资源  $r_j$  上的权重值
```

```
else //后继任务  $t_k$  不在同资源  $r_j$  上
   $rank_{up1} = rank_{up}(t_k) + w_{i,j} + c_{i,k}$ ;
end if
if  $rank_{up} < rank_{up1}$ 
   $rank_{up} = rank_{up1}$ ;
//根据式(4),求得任务  $t_i$  在资源  $r_j$  上的上行权重值
end if
end for each
end if
if  $rank_{up}(t_i) > rank_{up}$ 
   $rank_{up}(t_i) = rank_{up}$ ; //根据式(4)计算获得各个资源上的
  //最小权重值,即为任务  $t_i$  的权重值
   $pred\_R(t_i) = r_j$ ;
end if
end for each
end for each
```

对图1和表1对应的 DAG 计算每个节点的初始权重值结果如表2所示。

表2 HDPTS 计算的节点上行权重值及预期调度顺序

任务	$rank_{up}$	预期调度顺序	预分配资源	任务	$rank_{up}$	预期调度顺序	预分配资源
t_1	160	1	r_1	t_6	82	5	r_1
t_2	152	3	r_1	t_7	51	6	r_1
t_3	154	2	r_1	t_8	46	7	r_2
t_4	108	4	r_1	t_9	13	9	r_1
t_5	39	8	r_3	t_{10}	5	10	r_1

2.2.2 资源选择以及优化调度顺序

当全部任务入栈,即构造调度顺序表后,依次出栈。根据式(4)的计算特点,可以发现每个后继任务的上行权重值小于自身,又因为栈先进后出的特点,因此首先出栈的是入口节点,根据式(1)与式(2)选择使该节点完成时间最早的资源,该资源即为任务的调度资源。入口节点调度完毕后,继续出栈,优化那些所有前驱都已经调度完毕的任务,直到部分前驱没有调度的任务;然后按优化后的优先级权值从小到大入栈,再出栈调度;如此反复执行,直到所有任务调度完毕。IHEFT 算法仅仅是将分配之前的上行权重作为优先级确定任务的调度顺序,调度顺序在任务分配到具体资源之前就已经确定,这样的调度顺序并不理想;而本文算法在分配过程中根据任务分配到资源上的情况以及各个资源的负载不断优化调度顺序,这样的分配相对合理。优化任务调度优先级的计算方法如式(5)所示:

$$\begin{cases} priority(t_i) = rank_{up}(t_i) + \max_{r_j \in R} \{ \max_{t_j \in pred(t_i)} (AFT(t_j) + c_{ij}) \} \\ priority(t_{entry}) = rank_{up}(t_{entry}) \end{cases} \quad (5)$$

其中: t_i 任务已经完成调度; $avail[j]$ 表示调度到第 j 个资源上可以开始的调度时间,即已经调度到第 j 个资源上的最后一个任务的完成时间。

HDPTS 算法具体步骤如下:

- 步骤1 根据式(3)计算中每个任务所在的层次。
- 步骤2 用栈结构构造调度顺序表 pStack 将节点按层次逐层入栈。
- 步骤3 从 pStack 中依次出栈,根据式(4)计算节点的上

行权重值。

步骤4 将任务节点按上行权重值从小到大依次入pStack。

步骤5 判断pStack是否为空,如果不为空则转步骤6;如果为空,则调度结束。

步骤6 从pStack中出栈,判断是否为入口节点,是则转步骤13;否则转步骤7。

步骤7 节点优先级权值是否经过优化,如果没有优化则转步骤8;如果已经优化则转步骤13。

步骤8 判断这个任务节点的所有前驱是否调度完毕,如果是则转步骤9;否则转步骤11。

步骤9 根据式(5)优化任务节点优先级权重值,并将这个节点存放在一个临时队列pQueue中,然后判断pStack是否为空,如果不为空则转步骤10;如果为空则转步骤12。

步骤10 从pStack中出栈,转步骤8。

步骤11 将任务节点入pStack中,转步骤12。

步骤12 pQueue中的节点按优先级权值从由小到大进行排序并依次入pStack,转步骤6。

步骤13 将任务节点分配到使之完成时间最短的资源上,计算完成时间、调度资源,以及资源上的可以开始调度时间。

对图1与表1中对应的DAG,经过HDPTS优化调度顺序后的优先级值以及调度顺序见表3,调度顺序与之前通过上行权重计算优先级的调度顺序有所不同,主要是 t_7 与 t_8 的调度顺序发生改变,使调度顺序得到优化。使用HDPTS、IHEFT以及HEFT算法的调度结果如图2所示,使用HEFT与IHEFT算法的调度长度为210,而使用HDPTS算法的调度长度只有180,比使用IHEFT和IHEFT调度算法的调度长度减少14.29%,因此HDPTS算法的调度结果最好,主要得益于HDPTS算法在调度过程中动态优化调度顺序,使得调度顺序更为合理。

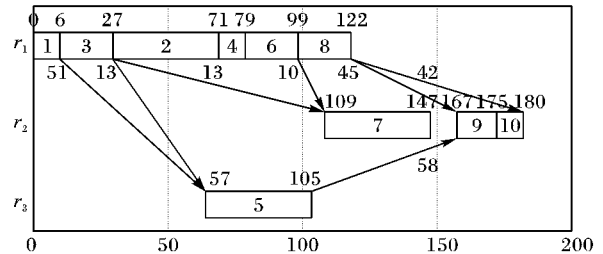
表3 HDPTS优化后的节点优先级权值及调度顺序

任务	优先级权值	调度顺序	分配资源	任务	优先级权值	调度顺序	分配资源
t_1	160	1	r_1	t_6	179	5	r_1
t_2	213	3	r_1	t_7	160	7	r_2
t_3	232	2	r_1	t_8	163	6	r_1
t_4	239	4	r_1	t_9	212	9	r_2
t_5	138	8	r_3	t_{10}	234	10	r_2

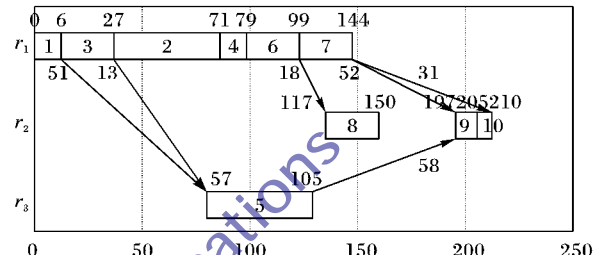
2.3 HDPTS算法性能分析

与其他类型的算法相比,表调度算法的复杂度比较低,HEFT算法与IHEFT算法的时间复杂度均为 $O(m \times e)$,其中 e 为DAG中边的数目, m 为资源数;当DAG中节点间通信频繁时,IHEFT算法的时间复杂度最大为 $O(m \times n^2)$ 。LDCP算法的时间复杂度为 $O(m \times n^3)$ 。本文算法主要由计算上行权重值和任务调度与调度优先级优化两部分构成,其中计算上行权重的复杂度为 $O(m \times e)$ 。任务调度即任务选择资源过程,时间复杂度为 $O(m \times e)$;任务调度优先级优化过程主要是前驱调度完后,将所有全部前驱调度完的后继出栈,进行调度优先级优化,然后进行排序,按从小到大优先级进行入栈,等待调度,开始任务不进行调度优先级优化,每个任务只进行一次调度优先级优化,即每个节点优先级只进行一次排序,这样最大时间复杂度为 $O(n^2)$ 。因此本文算法时间复杂度为 $O(n^2 +$

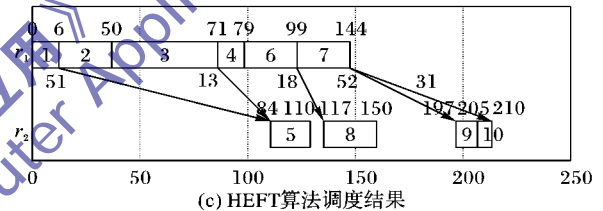
$m \times e)$,在任务间通信频繁或者资源较多的情况下,本文算法的时间复杂度为 $O(m \times n^2)$,并不比IHEFT算法时间复杂度高;如果资源较少且任务间通信较少,本文算法时间复杂度相对较高些。



(a) HDPTS算法调度结果



(b) IHEFT算法调度结果



(c) HEFT算法调度结果

图2 3种算法调度结果甘特图

DAG分布式任务调度问题一直是NP难问题,表调度算法在确定调度顺序阶段考虑全局信息,由于调度顺序在任务没有分配给具体资源时就已经确定,这样的调度顺序并不准确;而根据先前的确定的调度顺序分配到资源上,不能确定先分配的任务到哪个资源上,以及各个资源的负载情况,无法确定对后续任务分配的影响。算法IHEFT计算上行权重时是根据预定的资源进行计算,然后确定调度顺序,可是预定的资源与真实分配到的资源并不一样,这样确定的调度顺序显然也不准确;其次IHEFT算法没有考虑各个资源的负载情况,仅仅考虑了DAG本身,这也会影响调度顺序。HDPTS算法在IHEFT算法的基础上弥补了这些不足,首先通过上行权重值确定大致的调度顺序,然后在调度过程中根据先分配任务的情况以及资源负载情况优化后继调度顺序,边调度边优化调度顺序,实现调度顺序的动态优化,使得调度顺序更加精确、合理。

3 算法实验分析

3.1 实验环境

目前网络上公用的DAG算法仿真平台较少,文献中提到了名为EasyGrid的网络任务调度工具,但此工具并未真正公开,为了研究异构资源调度算法,本文在Qt5.1.1平台上用C/C++语言开发了一个DAG仿真平台,该平台目前拥有四种调度算法,如图3所示。

该平台拥有一个DAG生成模块,可以根据用户要求随机生成符合要求的DAG;该平台还有算法调度结果分析模块,可以对大量随机生成的DAG的多种算法调度结果进行对比

分析,通过甘特图直观看到调度结果,还可以查看这些节点调度顺序和优先级值。本文就利用这个平台将 HDPTS 算法与 IHEFT、HEFT 和 LDCP 算法进行了对比分析。

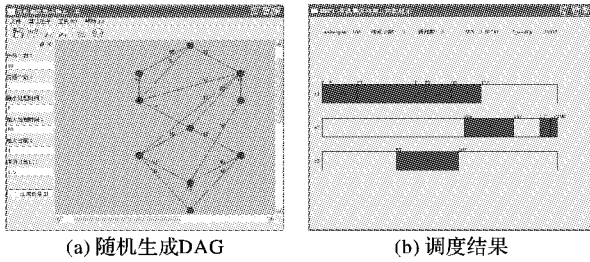


图3 仿真平台界面

3.2 参数及比较标准

本文利用四种算法对同一个 DAG 产生的调度结果进行对比和分析,本文描述 DAG 的主要参数有:1)任务节点个数 n ,即 DAG 中节点的个数;2)资源个数 m ,即调度环境中存在可用的资源;3)最大出度 \maxOutDegree ,即每个节点的出度不能超过 \maxOutDegree ;4)任务在资源上的最大处理时间 \maxProcTime 与最小处理时间 \minProcTime ;5)通信计算比 CCR ,即平均通信时间与平均计算时间之比。

在静态任务调度中,任务调度长度成为评价一个任务调度算法性能的重要标准,除此之外,还有调度长度比率、加速比等,具体如下:

- 1) 调度长度 $makespan$,为所有资源上的最大调度长度。调度长度越短,调度性能越好。
- 2) 调度长度比率 SLR ,计算公式如式(6)所示, CP_{min} 指当每个任务节点都使用最小计算时间的资源时所确定的关键路径上任务节点的集合,因此 SLR 总是大于 1。

$$SLR = makespan / \sum_{i \in CP_{min}} \min_{j \in R} \{w_{i,j}\} \quad (6)$$

3) 加速比 $Speedup$,计算公式如式(7)所示, $Speedup$ 越大表明调度算法性能越好。

$$Speedup = \frac{1}{makespan} \min_{i \in R} \left\{ \sum_{j \in R} w_{i,j} \right\} \quad (7)$$

4) 最优率 $BestRatio$,即进行 N 次测试后,某种算法取得最小 $makespan$ 的次数与 N 的比值。

3.3 实验与实验结果对比

本文通过四组不同的实验对比 HDPTS、IHEFT、HEFT 以及 LDCP 四种算法的性能。

实验 1 $m = 8$, $\minProcTime = 5$, $\maxProcTime = 50$, $\maxOutDegree = 4$, $CCR = 2.5$, $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$,进行 100 次实验,即随机生成 100 个 DAG 进行测试,实验获得的平均 SLR 、 $Speedup$ 和最优率如图 4 所示。通过实验数据对比分析发现,HDPTS 算法的 SLR 值最小,而 $Speedup$ 值最大;从图中还可以发现无论 n 多大,HDPTS 算法获得最优次数始终是最多的。

实验 2 $n = 100$, $\minProcTime = 5$, $\maxProcTime = 50$, $\maxOutDegree = 4$, $CCR = 2.5$, $m \in \{2, 4, 8, 16, 32\}$,进行 100 次实验所获得的平均 SLR 、平均 $Speedup$ 以及 $BestRatio$ 如图 5 所示。通过图形可以发现资源数量为 2 时,几种算法的调度结果相近;当资源数量达到一定数量后,HDPTS 算法调度结果明显优于其他算法。这主要得益于 HDPTS 算法调度过程中动态优化调度顺序,考虑了调度过程中资源的负载情况以及个任务的调度情况;随着资源数量进一步增加,HDPTS 算法相对于其他算法的优势减弱,但依然拥有优势。

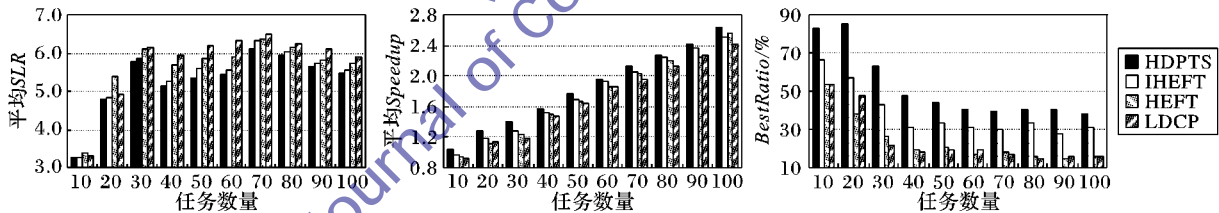


图4 不同任务数时各算法性能对比

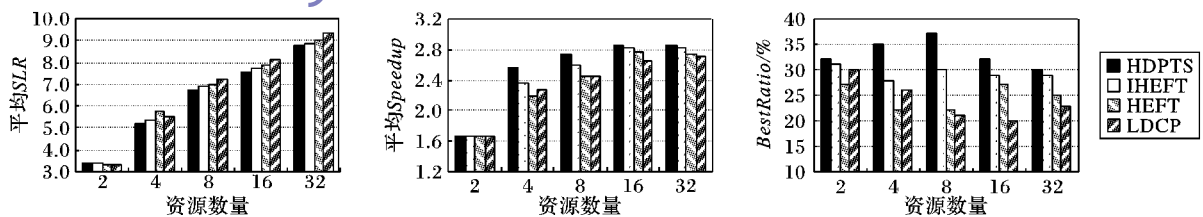


图5 不同资源数时各算法性能对比

实验 3 $n = 100$, $m = 10$, $\minProcTime = 5$, $\maxProcTime = 50$, $CCR = 16$, $\maxOutDegree \in \{2, 4, 6, 8, 10\}$,进行 100 次实验获得结果如图 6 所示。从中可以看出,当出度较小时各算法调度性能相近;随着出度增加,HDPTS 算法调度性能相对于其他算法的优势更明显,这说明 HDPTS 算法适用于并行度比较高的 DAG 调度;当出度为 10 时,HDPTS 算法优势有所减弱,但依然明显。

实验 4 $n = 100$, $m = 10$, $\minProcTime = 5$, $\maxOutDegree = 4$, $\maxProcTime = 50$, $CCR \in \{0.1, 0.5, 1.0, 1\}$,进行 100 次实验获得实验结果如图 7 所示。从中可以看

出, CCR 值较小时,各算法的调度性能相近;随着 CCR 增大,HDPTS 算法的调度性能明显优于其他算法。

通过实验可知 HDPTS 算法比 IHEFT 等算法更有效,主要是因为 HDPTS 算法在 IHEFT 算法的基础上,在调度过程中考虑了资源的负载情况,以及已经完成调度的任务对未调度任务的影响,综合这些因素,在调度过程中优化了后续调度任务的顺序;而 IHEFT 等算法对 DAG 的调度顺序在调度前就已经决定,使得误差较大,尤其是面对复杂的情况,比如节点出度与通信量较大、资源较多时,但 HDPTS 算法仍然可以取得较好的调度结果。

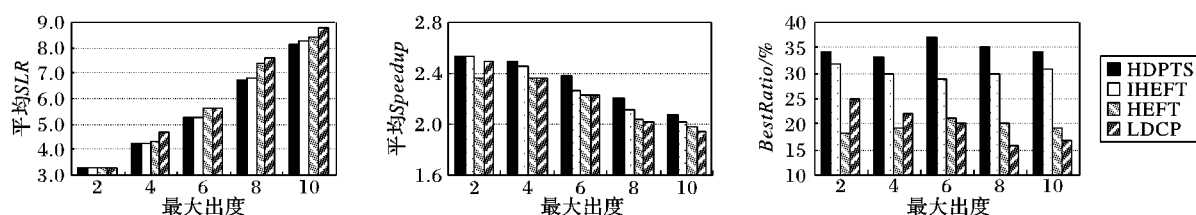


图6 DAG节点不同最大出度下各算法性能对比

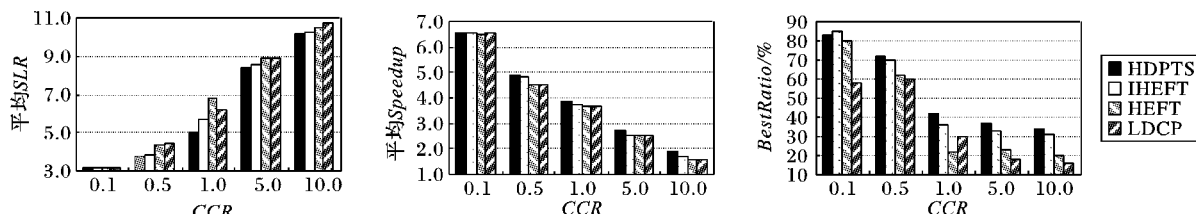


图7 不同通信计算比下各算法性能对比

4 结语

物联网是一种存在各种性能不同的资源的异构处理环境,高效的调度算法有利于提高任务处理效率,从而提高系统性能。表调度算法复杂度低,易于实现,但现有的表调度算法都是在整个任务系统调度前确定任务节点的调度顺序,无法准确地确定调度顺序,本文提出的HDPTS算法在调度过程中不断优化后继任务调度顺序,使得调度顺序更为合理,实验结果也表明与IHEFT、HEFT、和LDCP相比,HDPTS算法能取得较好的调度结果。采用表调度算法时,各个资源上存在大量的空闲时间,接下来会进一步考虑如何合理使用基于复制调度算法对HDPTS算法调度结果作进一步优化以取得更好的调度结果。

参考文献:

- [1] ATZORI L, IERA A, MORABITO G. The Internet of things: a survey [J]. *Computer Networks*, 2010, 54(15): 2787-2805.
- [2] TANG X, LI K, LI R, *et al.* Reliability-aware scheduling strategy for heterogeneous distributed computing systems [J]. *Journal of Parallel and Distributed Computing*, 2010, 70(9): 941-952.
- [3] MENG X, LIU W. A DAG scheduling algorithm based on selected duplication of precedent tasks [J]. *Journal of Computer-Aided Design and Computer Graphics*, 2010, 22(6): 1056-1062. (孟宪福, 刘伟伟. 基于选择性复制前驱任务的DAG调度算法[J]. 计算机辅助设计与图形学学报, 2010, 22(6): 1056-1062.)
- [4] YIN J, GU G, ZHAO J. Dynamic scheduling algorithm for hybrid real-time tasks with precedence constraints [J]. *Computer Integrated Manufacturing Systems*, 2010, 16(2): 411-416, 422. (殷进勇, 顾国昌, 赵靖. 优先约束的混合实时任务动态调度算法[J]. 计算机集成制造系统, 2010, 16(2): 411-416, 422.)
- [5] XIE Z, XIN Y, YANG J. Machine-driven integrated scheduling algorithm with rollback-preemptive [J]. *Acta Automatica Sinica*, 2011, 37(11): 1332-1343. (谢志强, 辛宇, 杨静. 可回退抢占的设备驱动综合调度算法[J]. 自动化学报, 2011, 37(11): 1332-1343.)
- [6] LAN Z, SUN S. An algorithm of allocating tasks to multiprocessors based on dynamic critical task [J]. *Journal of Computers*, 2007, 30(3): 454-462. (兰舟, 孙世新. 基于动态关键任务的多处理器任务分配算法[J]. 计算机学报, 2007, 30(3): 454-462.)
- [7] TOPCUOGLU H, HARIRI S, WU M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2002, 13(3): 260-274.
- [8] DAOUD M I, KHARMA N. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems [J]. *Journal of Parallel and Distributed Computing*, 2008, 68(4): 399-409.
- [9] WANG X, HUANG H, DENG S. List scheduling algorithm for static task with precedence constraints for cyber-physical systems [J]. *Acta Automatica Sinica*, 2012, 38(11): 1870-1879. (王小乐, 黄宏斌, 邓苏. 处理顺序约束的信息物理融合系统静态任务表调度算法[J]. 自动化学报, 2012, 38(11): 1870-1879.)
- [10] LI J, ZHANG B, WANG X. Heterogeneous multiprocessor task scheduling algorithm based on PSO [J]. *Application Research of Computers*, 2012, 29(10): 3621-3624. (李静梅, 张博, 王雪. 基于粒子群优化的异构多处理器任务调度算法[J]. 计算机应用研究, 2012, 29(10): 3621-3624.)
- [11] LI J, ZHANG B. Heterogeneous multiprocessor task scheduling based on PSO algorithm [J]. *Computer Engineering and Design*, 2013, 34(2): 2622-2624. (李静梅, 张博. 基于粒子群优化算法的群异构多处理器任务调度[J]. 计算机工程与设计, 2013, 34(2): 2622-2624.)
- [12] LI J, JIN S. Research on static scheduling based on heterogeneous multi-core processors [J]. *Computer Engineering and Design*, 2013, 34(1): 178-184. (李静梅, 金胜男. 基于异构多核处理器的静态任务调度研究[J]. 计算机工程与设计, 2013, 34(1): 178-184.)
- [13] LEE Y C, ZOMAYA A Y. A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2008, 19(9): 1215-1223.
- [14] YANG J, XU H, PAN L, *et al.* Task scheduling using Bayesian optimization algorithm for heterogeneous computing environments [J]. *Applied Soft Computing*, 2011, 11(4): 3297-3310.
- [15] MENG X, WANG M. Peer to peer task scheduling based on improved immune clonal selection algorithm [J]. *Computer Integrated Manufacturing Systems*, 2009, 15(9): 1795-1802. (孟宪福, 王敏. 基于改进免疫克隆选择的对等网络任务调度机制[J]. 计算机集成制造系统, 2009, 15(9): 1795-1802.)
- [16] TANG X, LI K, LIAO G, *et al.* A stochastic scheduling algorithm for precedence constrained tasks on grid [J]. *Future Generation Computer Systems*, 2011, 27(8): 1083-1091.