基于优先级表的实时调度算法及其实现*

王永炎+, 王 强, 王宏安, 金 宏, 戴国忠

(中国科学院 软件研究所 人机交互技术与智能信息处理实验室,北京 100080)

A Real-Time Scheduling Algorithm Based on Priority Table and Its Implementation

WANG Yong-Yan+, WANG Qiang, WANG Hong-An, JIN Hong, DAI Guo-Zhong

(Intelligent Engineering Laboratory, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62559307 ext 8817, E-mail: wyy@iel.iscas.ac.cn, http://iel.iscas.ac.cn

Received 2003-03-24; Accepted 2003-10-08

Wang YY, Wang Q, Wang HA, Jin H, Dai GZ. A real-time scheduling algorithm based on priority table and its implementation. *Journal of Software*, 2004,15(3):360~370.

http://www.jos.org.cn/1000-9825/15/360.htm

Abstract: This paper proposes a new scheduling scheme based on priority table design by integrating two characteristic parameters (i.e. deadline and value) of a task. Two real-time scheduling algorithms from the scheme are presented: earliest deadline value (EDV) and value earliest deadline (VED). Furthermore, how to implement the two algorithms using multi-linked lists is given, including task acceptance policy and task completion/abortion policy. This scheme can also be applied to integrate two other characteristic parameters or even three characteristic parameters of a task. Based on hit value ratio, weighted guarantee ratio and differentiated guarantee ratio, the performance of the VED and EDV algorithms are analyzed, the experimental results show that the VED and EDV algorithms can improve the performance compared to the classical EDF (earliest deadline first) and HVF (highest value first) algorithms under all workload conditions.

Key words: real-time scheduling; task; priority; hit value ratio; deadline guarantee ratio

摘 要: 讨论了综合考虑任务的截止期和价值两个特征参数的优先级表设计方法,提出了 EDV(earliest deadline value)与 VED(value earliest deadline)两种不同的基于优先级表的实时任务调度算法,并且利用多重链表给出了这两种算法的实现,包括任务接收策略与任务完成/夭折策略的算法实现.这种优先级表设计方法及其基于多重链表的实现方法也适用于对任务的其他两种甚至 3 种不同特征参数之间的综合.基于累积实现价值率、加权截止期保证率与差分截止期保证率 3 个方面,分析了 VED 算法与 EDV 算法的性能,实验结果表明,在所有负载条件下 VED 算法与 EDV 算法相对于 EDF(earliest deadline first)算法与 HVF(highest value first)算法都

* Supported by the National Natural Science Foundation of China under Grant Nos.60373055, 60374058 (国家自然科学基金)

作者简介: **王永炎**(1978－),男,福建永定人,博士生,主要研究领域为实时系统,实时智能,故障诊断;**王强**(1972－),男,博士生,主要研究领域为实时系统,实时数据库,软件工程;**王宏安**(1963－),男,博士,研究员,博士生导师,主要研究领域为实时智能系统,实时调度,人机交互;**金宏**(1978－),男,博士,副研究员,主要研究领域为实时系统,智能调度,控制设计;**戴国忠**(1944－),男,研究员,博士生导师,主要研究领域为人机交互技术,实时智能,软件工程.

有很大的性能改进.

关键词: 实时调度;任务;优先级;实现价值率;截止期保证率
中图法分类号: TP316 **文献标识码**: A

基于优先级的调度算法在实时系统中有着广泛的应用,这类算法主要包括单调速率(rate-monotonic,简称RM)算法[1]、截止期最早最优先(earliest deadline first,简称 EDF)[1]和空闲时间最短最优先(least slack first,简称 LSF)、价值最高最优先(highest value first,简称 HVF)[2]、价值密度最大最优先(highest value density first,简称 HVDF)[2,3]等等.在这些算法中,任务的优先级都是基于任务的某些特征参数,如截止期、空闲时间或关键性(即任务的重要程度或者价值)等计算而得.然而,如果优先级仅由某个特征参数来确定是不够的[4~6].如 EDF 策略是将最高优先级指派给具有最早截止期的任务,LSF 策略是将最高优先级指派给具有最短空闲时间的任务,尽管在正常的系统负载下这些算法表明了其最优性,但是在过载的情况下,系统不可能保证所有的任务都能够满足截止期,这时 EDF 或者 LSF 算法会出现急剧的性能降级,甚至导致多米诺现象[3].再者,截止期或者空闲时间短的任务不一定是最关键的,但是即使在过载的情况下系统也应该保证关键任务的及时完成,从而支持系统性能优雅地降级,不致出现系统失效甚至崩溃.基于价值的调度算法,如 HVF,能够通过优先执行具有最大价值或者最关键的任务来提供系统在过载情况下的性能降级.基于价值的调度算法的性能指标通常采用系统实现的累积价值来衡量,因此人们进一步提出了 HVDF 算法,优先调度执行价值密度(value density,简称 VD,任务的价值与其估计执行时间的比值)最高的任务.无疑,HVDF 算法是一种贪婪的算法,总是假定具有较高价值密度与较小空闲时间的任务将很快到达,因此执行具有最高价值密度的任务而较早地累积价值,往往导致许多任务不必要地错失截止期.

许多研究[3,7]表明,优先级驱动的调度算法应该综合任务的关键性与截止期两个独立的特征参数.例如,最大努力(best effort,简称 BE)算法基于任务的价值密度来接纳任务,但是按照 EDF 策略调度这些任务,尽管这种算法在不考虑算法复杂性与调度开销的情况下优于基于单特征参数的调度算法,如 EDF 或 LSF,但是文献[8]所进行的实验结果表明,BE 算法在系统过载的情况下会有非常大的系统开销.文献[7]提出了关键性-截止期优先(criticalness-deadline first,简称 CDF)算法,每个任务在到达时基于(相对截止期÷关键性)分配优先级,实验表明,基于 CDF 的 CPU 调度,相对于单独使用截止期或者关键性作为特征参数的算法,能够很大地改进系统的综合性能.文献[3]对 4 种调度策略:EDF,HVF,HVDF 与 MIX 进行了综合仿真研究,其中 MIX 算法的优先级 $p_i=(\alpha)V_i-(1-\alpha)d_i$,$p_i$,$V_i$ 与 d_i 分别表示任务 i 的优先级、价值与截止期,这 4 种策略被进一步扩展,以两种方式管理过载,或者简单地拒绝到来的任务或者移去最小价值的任务,直到排除过载为止.研究的结论是,在过载前按截止期调度,在过载期间按价值调度,这样,在大多数实际情况下表现最好.文献[9,10]描述了一个综合离线调度与基于价值的动态调度的运行时调度方法,要求所有定期任务必须离线地调度,而通过过载检测方法来防止动态到达的非定期任务导致系统过载,并把过载处理问题形式化为二元最优化问题.这个算法与 BE 算法类似,会有较大的运行时开销.

总的来说,上述算法存在以下不足之处:(1) 由于任务的截止期与价值是两个完全不同的概念,其取值范围也不同,因此不能简单地将它们进行加权运算;(2) 由于操作系统或者调度系统通常只支持有限的优先级,CDF 与 MIX 算法所计算的优先级必须转换到系统所支持的优先级层次上,因此会导致不同的任务具有相同的优先级;(3) BE 之类的启发式算法具有较大的系统开销,特别是在系统过载的情况下,较高的过载处理开销会更加严重地影响系统性能.为了克服这些问题,本文提出了基于优先级表的实时调度算法,其中任务的优先级不再通过对其特征参数进行算术运算来分配,而是通过排序充分体现任务的特征参数之间的序列关系.这种算法不仅能够用来综合任务的价值与截止期两个特征参数,而且同样适用于在调度中综合考虑任务的其他任何两个特征参数,甚至综合任务的 3 个不同的特征参数.本文重点讨论了综合任务的截止期与价值的优先级表设计方法,提出了 EDV(earliest deadline value)与 VED(value earliest deadline)两种调度算法,给出了基于多重链表的算法实现,并进行了仿真实验与分析.

1 任务模型

在描述具体的调度算法之前,首先给出任务及其参数的定义与表示.假设存在任务 J_i,具有下面的参数:

- a_i 表示任务的到达时间,即任务被启动并准备执行的时间;
- C_i 表示任务的最坏情形执行时间,即任务在最坏情况下无中断执行所需的处理器时间;
- c_i 表示任务的剩余执行时间,即任务的最坏情况执行时间与任务已经执行时间之差;
- d_i 表示任务的绝对截止期,即任务在这个时间应该完成执行并产生一个有价值的结果(在后面的讨论中,如果没有特别说明,我们所说的截止期都是指任务的绝对截止期);
- D_i 表示任务的相对截止期,有 $D_i=d_i-a_i$;
- V_i 表示任务的价值,即任务的关键性,该任务相对于任务集中其他任务的重要程度;
- f_i 表示任务的完成时间.

假设当任务到达时,任务表示为 $J_i=(C_i,D_i,V_i)$,其到达时间是预先未知的.本文后面的讨论只考虑任务具有固定截止期的情况,即任务一旦到达截止期,其价值立即降低为 0,并且任务的执行也将立即终止.调度算法的性能从 3 个方面被考察:(1) 实现价值率(hit value ratio,简称 HVR)是调度算法调度执行任务满足截止期所实现的累积价值与所有提交的任务价值总和的比率;(2) 加权截止期保证率(weighted guarantee ratio,简称 WGR)是指不同关键程度的任务的被满足截止期的情况,它体现了一个调度算法的健壮性;(3) 差分截止期保证率(differentiated guarantee ratio,简称 DGR),即使在系统严重过载的情况下,调度算法也必须尽量保证较高价值(即较重要)的任务满足截止期,因此,使用 DGR 来评估调度算法在系统过载时对于不同价值的任务所产生的差分服务能力.

2 基于优先级表的调度算法

优先级分派策略可看成是一个函数,它可以针对两类不同情况:单个任务或一个任务集.当用于单个任务时,函数的结果就是对应分派策略所确定的该任务的优先级;当用于任务集时,函数的结果是任务集中任务的一个排序表,在实施调度时,优先级最高者排第 1,最低者排最后.文献[11]在设计优先级表时,考虑的是任务的相对截止期和空闲时间两个特征参数,将任务的相对截止期和空闲时间这两个参数的取值范围划分成若干个不同的取值区间,每个区间选择一个典型值来代表这个区间.对具有不同的典型相对截止期 D_i 和典型空闲时间 s_j 的任务 J_{ij},赋予其特定的优先级值 $P_{ij}(d_i,s_j)$,从而得到事先能够确定的优先级表 $P=(P_{ij})$.对于具有任一相对截止期 D 和空闲时间 s 的任务,当 D 和 s 是典型截止期之一和典型空闲时间值之一时,则查表可获得该任务的优先级;当 D 和 s 中至少有一个不是典型值时,则通过对优先级表进行插值,以便获得该任务的优先级,文献[11]中给出了二元三点插值公式.

文献[11]中基于优先级表设计与插值方法的优先级分派策略通过综合任务的截止期与空闲时间这两个特征参数,有效地提高了任务调度的成功率.但是,算法要求必须事先明确特征参数的典型值并计算优先级表,而典型值的设置又会影响插值的效果.针对动态实时系统中的任务调度与过载处理问题,下面给出一种综合任务的截止期与价值的优先级表设计方法,不需要预先确定任务参数的典型值并计算优先级表,而是在线地为任务分配优先级,并按照优先级调度这些任务.

2.1 截止期/价值优先级表设计

截止期/价值优先级表设计方法的目标是在任务调度时综合考虑任务的截止期与价值,从而保证系统在过载情况下能够优雅地降级,不会出现 EDF 之类调度算法中存在的多米诺现象.首先我们来明确算法调度的原则:

- 任务的截止期越早且价值越大,则任务的优先级越高;
- 对于截止期与价值完全相同的任务,先到达者具有更高的优先级.

图 1 给出了基于截止期/价值的一种优先级方案表,其中箭头表示了任务的优先级顺序.在这种优先级分派方法中,任务的截止期序列按照升序排列,即截止期越早,任务越靠前;而任务的价值序列则是采用降序排列,即任务价值越大越靠前.对于具有同样截止期的任务,价值越大,任务的优先级越高;而对于具有同样价值的任务,

截止期越早,任务的优先级越高.

在如图 1 所示的优先级表中,每个任务具有一个优先级等级:

$$P=i+j,$$

其中 i 与 j 分别表示任务的截止期与价值在各自序列中的位置.图中每条斜线上标识的任务具有相同的优先级等级 P,但是同样优先级等级的任务,其调度执行的顺序必须按照箭头所指的方向,表现为不同的优先级.任务的优先级 p 可以按照下式计算:

$$p=(i+j-1)*(i+j-2)/2+i.$$

p 值越小,任务的优先级越高.对于具有相同优先级等级的任务,调度算法更加倾向于截止期较早的任务,即同一优先级等级中的任务,截止期较早的任务具有更高的优先级.这种优先级表设计模式给出了一种综合任务的截止期与价值的调度算法,我们称为 EDV 算法.

类似地,我们给出任务优先级表的另一种设计模式,如图 2 所示.对于具有相同优先级等级的任务,调度算法更加倾向于价值较大的任务.这种模式下,任务的优先级可以按照下式计算:

$$p=(i+j-1)*(i+j-2)/2+j.$$

基于这种模式的调度算法称为 VED 算法.

尽管任务的优先级是按照优先级表来分配,但是我们的方法并不事先计算优先级表,而是通过运行时动态组织任务的优先级表来达到基于优先级的任务调度.

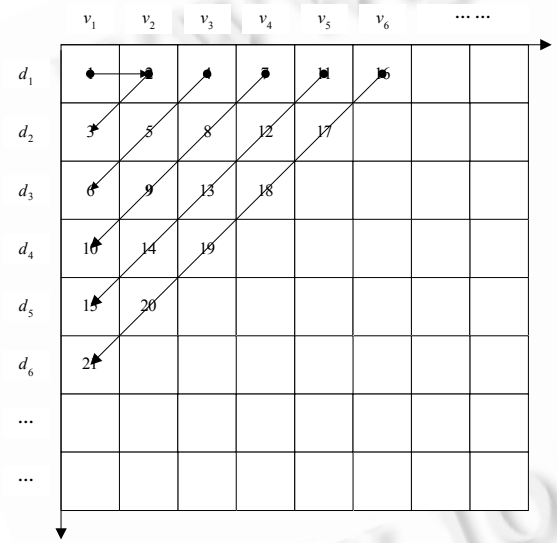


Fig.1 EDV priority table design

图 1 EDV 优先级表设计

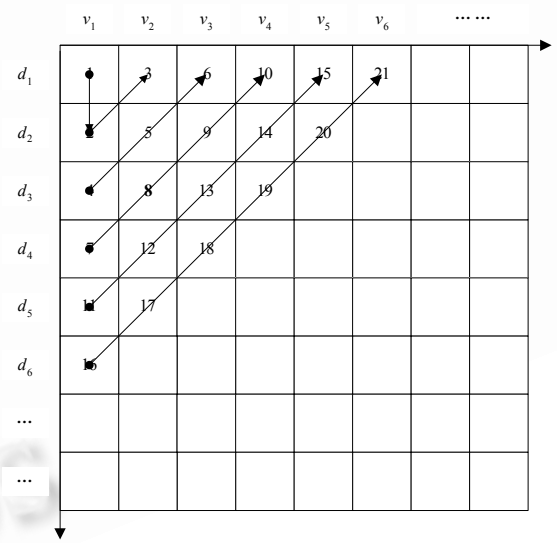


Fig.2 VED priority table design

图 2 VED 优先级表设计

2.2 优先级表的加权方法

上面提出的 EDV 与 VED 算法分别倾向于截止期较早与价值较大的任务,下面我们讨论如何通过加权方法加强这种倾向性.优先级表加权方法引入了一个权重参数 γ,γ 为正整数.同样,优先级表加权方法也分为两种:倾向截止期的加权方法(weighted-EDV,简称 WEDV)和倾向价值的加权方法(Weighted-VED,简称 WVED).在倾向截止期的加权方法中,任务优先等级 P 和优先级 p 的计算公式为

$$P=(\gamma*i-\gamma+1)+j,$$

$$p=(\gamma*(i-1-u)+2*j-2)*(i+u)/2+i.$$

其中 $u=\lfloor (j-2)/\gamma \rfloor$.当$\gamma=1$ 时,该算法就是 EDV 算法.当$\gamma=2$ 时,任务的优先级表如图 3 所示.当γ取值足够大时,该算法相当于 EDF 算法.

在倾向价值的加权方法中,任务优先等级 P 和优先级 p 的计算公式如下:

$$P=i+(\gamma*i-\gamma+1),$$

$$p=(\gamma*(j-1-u)+2*i-2)*(j+u)/2+j.$$

其中 $u=\lfloor(i-2)/\gamma\rfloor$.当$\gamma=1$ 时,该算法就是 VED 算法.当$\gamma=2$ 时,任务的优先级表如图 4 所示.当γ取值足够大时,该算法相当于 HVF 算法.

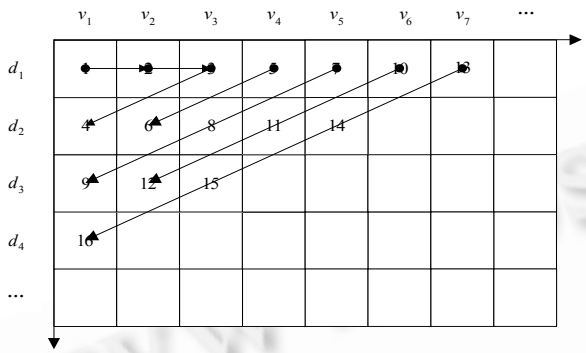


Fig.3 WEDV priority table design

图 3 WEDV 优先级表设计

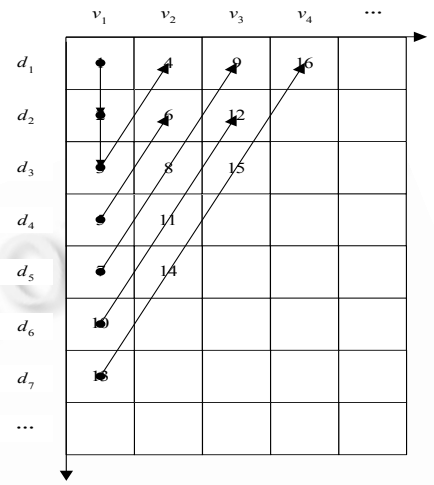


Fig.4 WVED priority table design

图 4 WVED 优先级表设计

2.3 非方阵优先级表设计

假设任务的价值根据其取值范围被划分为 N 个区间或者任务具有 N 个不同关键等级,则优先级表的价值维是固定的,因此优先级表为非方阵结构.在非方阵优先级表中,任务优先级等级 P 和优先级 p 的计算公式如下:

$$P=i+n,$$

$$p=\begin{cases}(i+n-1)*(i+n-2)/2+i, & P\le N+1\\ N*(N+1)/2+(i+n-N-1)*N-n+1, & P>N+1\end{cases}.$$

其中 n 为任务所在的区间号.图 5 给出了 $N=5$ 时 EDV-N 算法的优先级表.

2.4 三维优先级表设计

优先级表方法能够被进一步扩展,在调度中集成任务的 3 个不同的特征参数.图 6 给出了一种综合任务的截止期、松弛时间与价值的三维优先级表设计模式,其中,截止期序列按照升序排列,松弛时间序列也按照升序排列,而价值序列按照降序排列.处于同一斜剖面上的任务属于同一优先级等级:

$$P=i+j+k.$$

其中,i,j 与 k 分别表示任务在截止期、松弛时间与价值序列中的位置.同一优先级等级上的任务,由于每个特征参数在任务优先级分配中比重不同,仍然使得每个任务具有不同的优先级:

$$p=(P-1)*(P-2)*(P-3)/6+(2*P-i-2)*(i-1)/2+j.$$

在上面的三维优先级表设计模式中,任务的 3 个特征参数的重要程度由高到低依次为:截止期、松弛时间与价值.

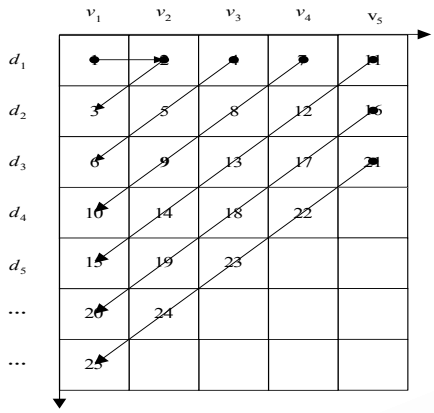


Fig.5 EDV-5 none-square matrix priority table design

图 5 EDV-5 非方阵优先级表设计

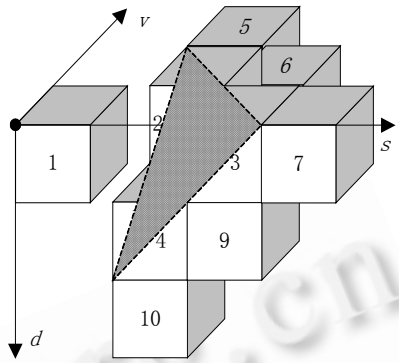


Fig.6 Three-Dimension priority table design

图 6 三维优先级表设计

这种优先级表设计模式,不仅可以通过偏重于不同的特征参数来实现不同的调度算法,而且能够用于综合任务的其他任何 3 个特征参数.

3 实现基于优先级表的调度算法

本节以 EDV 算法为例,给出基于优先级表的调度算法的一种实现.基于优先级表的特点,任务的截止期序列与价值序列分别实现为基于截止期的任务链表 Q^d 与基于价值的任务链表 Q^v,Q^d 与 Q^v 都是带空闲头结点的双向循环链表,交错形成一个逻辑上的优先级表.之所以使用双向链表,是为了便于任务结点的插入与移除.

任务结点 TaskNode 的结构与链表的定义如下:

```
typedf struct TaskNode{
    int a;   //任务的达到时间
    int c;   //任务的剩余执行时间
    int d;   //任务的截止期
    int v;   //任务的价值
    int i,j; //任务的行与列下标
    int p;   //任务的优先级
    struct TaskNode *pdprior, *pdnext;   //基于截止期的任务链表指针
    struct TaskNode *pvprior, *pvnext;   //基于价值的任务链表指针
}TaskNode, *TaskLink
TaskLink dhead;
TaskLink vhead;
TaskNode *pactive
```

其中,dhead 与 vhead 分别是 Q^d 与 Q^v 的表头结点,pactive 指向当前正在执行的任务结点.

在系统运行过程中,当新的任务到达、任务完成或者任务夭折时,需要调整优先级表,并且确定当前最高优先级的任务去执行.基于这样的优先级表组织结构,我们给出了线性时间复杂性的任务接收策略与任务完成/夭折策略来进行新任务的接收与任务的完成/夭折处理.

3.1 任务接收策略

当新的任务到达时,系统调用任务接收策略,把任务插入 Q^d 与 Q^v,并计算出任务的优先级,判断是否需要抢占当前正在执行的任务.

任务接收策略的算法描述如下:

1) 设 *pnew* 指向新任务,其中指针域初始为 *Null*;
2) 把任务 *pnew* 插入 Q^d
   (a) *pd=dhead->pdprior*; //定义指针,指向 Q^d 的尾结点
   (b) *i=pd->i*+1; //初始化新任务在优先级表中的行下标
   (c) while ((*pd*!=*dhead*) and (*pnew->d<pd->d*)) { //确定新任务在 Q^d 中的位置
   (d) *pd->i*++; *pd=pd->pdprior*; *i*−−}
   (e) *pnew->i=i*; //设置新任务在优先级表中的行下标
   (f) insert(*pd*,*pnew*); //把 *pnew* 插入 *pd* 之后
3) 把任务 *pnew* 插入 Q^v
   (a) *pv=vhead->pvprior*; //定义指针,指向 Q^v 的尾结点
   (b) *j= pv->j*+1; //初始化新任务在优先级表中的列下标
   (c) while ((*pv*!=*vhead*) and (*pnew->v>pv->v*)) { //确定新任务在 Q^v 中的位置
   pv->j++; *pv=pv->pvprior*; *j*−−}
   (d) *pnew->j=j*; //设置新任务在优先级表中的列下标
   (e) insert(*pv*,*pnew*); //把 *pnew* 插入 *pv* 之后
4) 计算任务的优先级并判断是否需要抢占当前正在执行的任务
   (a) *pnew->p*=(*i*+*j*−1)*(*i*+*j*−2)+*i*;
   (b) if (*pnew->p>pactive->p*) { //新任务 *pnew* 抢占当前任务
   pactive; *pactive=pnew*;}

这个算法分别从两个链表的尾结点开始逆向查找,以确定新的任务结点在优先级表中的位置与其优先级.在最坏情况下,需要对 Q^d 与 Q^v 各扫描一遍,因此算法的复杂度为 $O(2n)$,其中 n 为当前优先级表中的任务数.

3.2 任务完成/夭折策略

当一个任务完成执行或者超过截止期而夭折时,系统需要调用任务完成/夭折策略,把完成执行或者夭折的任务结点从优先级表中移除,并且相应地调整后续任务的行与列下标以重新计算任务的优先级,并且选择优先级最高的任务去执行.

任务完成/夭折策略的算法描述如下:

1) 从优先级表中移除当前完成的任务
   (a) //从 Q^d 链表中移除该任务结点
   pd=pactive->pdprior->pdnext=pactive->pdnext;
   pactive->pdnext->pdprior=pactive->pdprior;
   (b) //从 Q^v 链表中移除该任务结点
   pv=pactive->pvprior->pvnext=pactive->pvnext;
   pactive->pvnext->pvprior=pactive->pvprior;
   (c) delete *pactive*; *pactive=Null*;
2) Q^d 链表中所有后续任务的行下标减 1
   while(*pd*!=*dhead*) {
   pd->i−−; *pd->p*=(*pd->i*+*pd->j*−1)*(*pd->i*+*pd->j*−2)/2+*i*;
   pd=pd->pdnext;}

3) Q^v链表中所有后续任务的列下标减 1

```
while(pv!=vhead) {
    pv->j--; pv->p=(pv->i+pv->j-1)*(pv->i+pv->j-2)/2+i;
    pv=pv->pvnext;}
```

4) 从 Q^d链表头结点开始扫描确定优先级最高的任务去执行

```
(a) pd=pactive=dhead->pdnext;
(b) hp=0; //记录当前扫描过的任务中的最高优先级
    if (pactive!=dhead) {
       hp=pactive->p; pd=pd->pdnext;}
(c) while(pd!=dhead) {
        if (pd->p>hp) {hp=p; pactive=pd;}
        pd=pd->pdnext;
      }
(d) if (hp!=0)   //任务 pactive 占用处理器并开始执行
```

这个算法首先从链表 Q^d与 Q^v中移除给定的任务结点,并且从被移除结点的下一个结点开始更新任务的行与列下标并重新计算任务的优先级,这在最坏情况下需要对 Q^d 与 Q^v 各扫描一遍.确定最高优先级任务的过程是扫描 Q^d链表(当然也可以扫描 Q^v链表),时间复杂度为 $O(n)$.因此算法总的复杂度为 $O(3n)$,其中 n 为当前优先级表中的任务数.

4 性能分析

为了评估 EDV 算法与 VED 算法在不同负载下的性能,我们以 EDF 算法与 HVF 算法作为基线,从实现价值率 HVR、加权截止期保证率 WGR 与差分截止期保证率 DGR 这 3 个方面比较 EDV 算法、VED 算法相对于 EDF 算法与 HVF 算法在性能方面所具有的优越性,以及这些算法的优缺点.

在我们做的所有仿真实验中,任务集由 100 个任务 $J_i(i=1,2,3,\ldots,100)$组成,任务的参数根据下面的方法产生:

(1) 任务的最坏情形执行时间 C_i在 5~105 个时间单元之间随机选择,服从均匀分布;

(2) 任务的每个实例 J_{ij}到达的时间服从均值为 T_i的指数分布,T_i按照下面的公式计算:$T_i=N*C_i/\rho$,其中 N 表示任务集中的总任务数(此处为 100),ρ表示期望产生的工作负载,称为标称负载;

(3) 设f^s表示任务的松弛时间与最坏情形执行时间的比率,并且f^s服从均值 2.0 的指数分布,使得任务实例 J_{ij}的相对截止期 $D_{ij}=C_i+f^s{}_j*C_i$;

(4) 由于任务的实际执行时间通常小于任务的最坏情形执行时间,因此定义f^e表示任务的实际执行时间与最坏情形执行时间的比率,并且f^e服从 0.4~1.0 之间的均匀分布,则任务实例的平均实际执行时间为 $0.7*C_i$;

(5) 任务的价值 V_i服从 1~100 之间的均匀分布,并且所有任务被划分到关键程度不同的 $k(0\le k\le 9)$个类别中,其中 k 越大表示该类别任务的关键程度越高,任务 J_i属于第 k 类当且仅当 $k*10<V_i\le(k+1)*10$.

本文所有的实验结果都是采用 100 次独立实验结果的平均值,每次实验运行的持续时间为 30 000 个时间单元.标称负载ρ是基于任务的最坏情形执行时间,为了比较算法在不同负载情形下的性能,实验中,ρ的取值范围为 0.5~3.5.

4.1 实现价值率

图 7 给出了 EDF 算法、HVF 算法、EDV 算法与 VED 算法在不同负载下的 HVR 变化趋势,其中 x 轴是标称负载ρ,y 轴表示不同调度算法的实现价值率 HVR.

从图中我们不难发现,EDF 算法在负载较低的情形下,表现出了其最优性,如ρ=0.5 时,HVR 接近 100%.但是,随着负载的不断增加,EDF 的性能急剧下降,相对而言,其他算法能够更加优雅地降级.HVF 算法倾向于优先执行价值最大的任务,但是调度中没有考虑任务的截止期,因此易于导致任务错失截止期,即使在较低的负载

下,HVR 也是所有算法中最低的.但是随着负载的增加,特别是在ρ>2.0 以后,这种算法表现出比 EDF 更好的性能.无论如何,综合考虑任务的截止期与价值的 EDV 与 VED 算法在几乎所有的负载情况下都表现出了比 EDF 算法和 HVF 算法更好的性能.

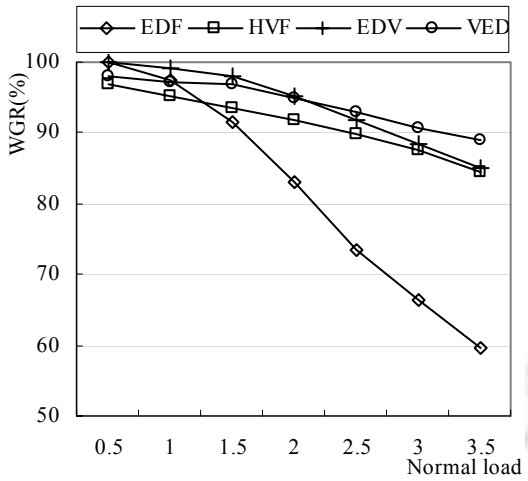


Fig.7 Hit value ratio

图 7 实现价值率

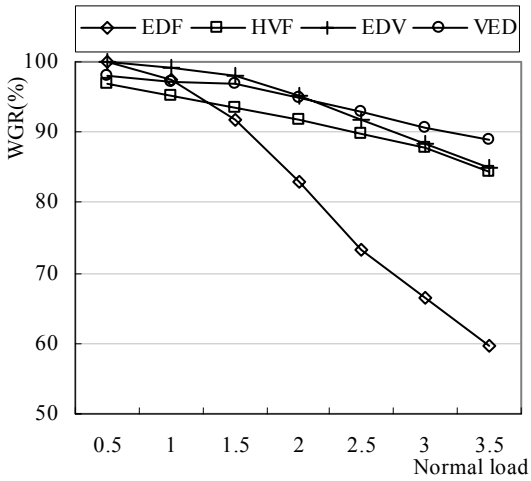


Fig.8 Weighted guarantee ratio

图 8 加权保证率

EDV 算法与 VED 算法相比,在较低的负载下 EDV 算法优于 VED 算法,但是当标称负载达到 2.0 以后,VED 算法表现出更好的性能,其原因在于 VED 算法更加偏重于考虑任务的价值,因此能够在较高的负载下实现更高的累积价值.

4.2 加权截止期保证率

加权截止期保证率(weigted guarantee ratio,简称 WGR)是指不同关键程度的任务的被满足截止期的情况,它体现了一个调度算法的健壮性.当然,这个值也与使用的加权系数有关,具体的加权值应该根据应用环境决定,这里我们利用公式

$$WGR = 100 * \frac{\sum_i (w_i * T_G^i)}{\sum_i (w_i * T_C^i)},$$

其中 i 表示任务的关键程度,$w_i=2^i$ 表示不同类别任务的权重,T_G^i 表示满足截止期的第 i 类任务的总数,T_C^i 表示提交的第 i 类任务的总数.

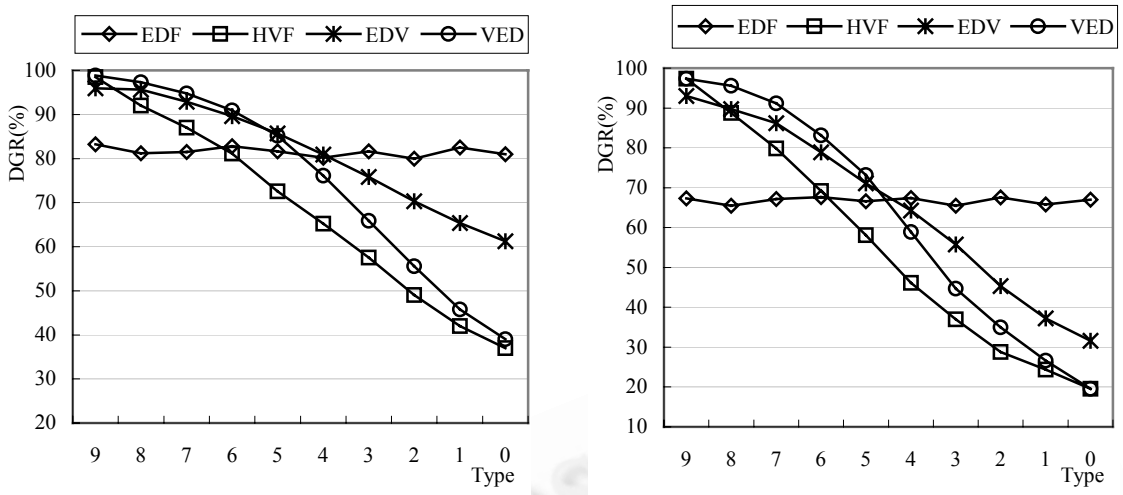
图 8 给出了 EDF 算法、HVF 算法、EDV 算法与 VED 算法在不同负载下的 WGR 变化趋势,其中 x 轴是标称负载ρ,y 轴表示调度算法在不同负载下的加权截止期保证率 WGR.

从图中我们不难发现,EDF 算法随着系统负载的增加其性能急剧下降,而其他算法能够较优雅地降级.由 EDV 算法与 VED 算法的比较,我们发现:在较低的负载下,EDV 算法表现出最好的性能;随着负载的增加,VED 算法逐渐表现出其优越性,当标称负载达到 2.5 以后,VED 算法明显优于其他算法.

4.3 差分截止期保证率

由于价值较大的任务相对重要,因此调度算法应该优先保证价值较大的任务满足截止期.这里,我们采用差分截止期保证率比较不同算法在系统过载时对于不同关键程度的任务所能够提供的差分服务能力.

在我们的实验中取标称负载ρ=2.0 与ρ=3.0 两种情况,分别代表了算法在一般过载与严重过载两种情况下所表现的性能.图 9(a)与图 9(b)分别给出了算法在这两种负载下对不同类别任务所提供的截止期保证率.



(a) ρ=2.0 (b) ρ=3.0

Fig.9 Differentiated guarantee ratio

图 9 差分保证率

由图 9 不难发现,EDF 算法不加区别地调度执行任务,因此所有任务在不同负载下具有差不多相同的截止期保证率,而其他算法则都具有一定的差分服务能力,能够为关键级别高的任务提供更好的服务.由于 HVF 算法总是优先调度执行关键程度最高的任务,然后是关键程度稍低的任务,依此类推,因此在 HVF 算法调度下任务的截止期保证率从最高关键程度的超过 95%,依次下降,呈线性关系.

在系统一般过载情况下,综合考虑任务截止期与价值的 EDV 与 VED 算法都能够为关键程度最高的 4 类任务提供超过或者接近 90%的截止期保证率;而在系统严重过载的情况下,EDV 与 VED 算法都能为关键程度最高的 4 类任务提供超过或者接近 80%的截止期保证率.并且,无论是在一般过载还是在严重过载情况下,VED 算法都能实现关键程度最高的 3 类任务的截止期保证率在 90%以上.

总体上讲,EDF 算法只考虑任务的截止期,为所有任务提供同样的截止期保证率,随着负载的增加会出现急剧的性能降级.HVF 算法总是优先执行较高价值类任务,尽管在较高负载情形下仍能为高价值类任务提供较好的截止期保证率,但是总体性能较差.实验结果表明,VED 算法与 EDV 算法相对于 EDF 算法与 HVF 算法都有很大的性能改进.

5 结 语

本文讨论了综合考虑任务的截止期和价值两个特征参数的优先级表设计方法,相对于先前研究中提出的综合任务截止期与价值的算法,无须限制截止期与价值的取值范围,也不需要对这两种不同的特征参数进行加权运算,并且保证了任务所分配的优先级具有惟一性.文中提出了 VED 与 EDV 两种不同的基于优先级表的实时任务调度算法,并且利用多重链表给出了这两种算法的实现,包括任务接收策略与任务完成/夭折策略的算法实现.这种优先级表设计方法及其基于多重链表的实现方法也适用于对任务的其他两种甚至 3 种不同特征参数之间的综合.另外,以 EDF 与 HVF 算法为基线,我们从累积实现价值率 HVR、加权截止期保证率 WGR 与差分截止期保证率 DGR 这 3 个方面对 VED 算法与 EDV 算法的性能进行了实验与分析,结果表明,VED 算法与 EDV 算法相对于 EDF 算法与 HVF 算法都有很大的性能改进.

References:

[1] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 1973, 20(1):46~61.

[2] Jensen ED, Locke CD, Toduda H. A time-driven scheduling model for real-time operating systems. In: Proc. of the 6th IEEE Real-Time Systems Symp. San Diego: IEEE Computer Society Press, 1985. 112~122.

[3] Buttazzo G, Spuri M, Sensini F. Value vs. deadline scheduling in overload conditions. In: Proc. of the 19th IEEE Real-Time Systems Symp. Pisa: IEEE Computer Society Press, 1995. 90~99.

[4] Biyabani SR, Stankovic JA, Ramamritham K. The integration of deadline and criticalness in hard real-time scheduling. In: Proc. of the 9th IEEE Real-Time Systems Symp. Huntsville: IEEE Computer Society Press, 1988. 152~160.

[5] Tseng S-M, Chin YH, Yang W-P. Scheduling value-based transactions in real-time main-memory databases. In: Lin KJ, ed. Proc. of the 1st Int'l Workshop on Real-Time Databases: Issues and Applications. Newport Beach: Kluwer Academic Publishers, 1996. 111~117.

[6] Burns A, Prasad D, Bondavalli A, Giandomenico FD, Ramamritham K, Stankovic J, Strigini L. The meaning and role of value in scheduling flexible real-time systems. Journal of Systems Architecture, 2000,46(4):305~325.

[7] Huang JD, Stankovic JA, Towesly D, Ramamritham K. Experimental evaluation of real-time transaction processing. In: Proc. of the 10th Real-Time Systems Symp. Santa Monica: IEEE Computer Society Press, 1989. 144~153.

[8] Wendorf JW. Implementation and evaluation of a time-driven scheduling processor. In: Proc. of the 9th IEEE Real-Time Systems Symp. Huntsville: IEEE Computer Society Press, 1988. 172~180.

[9] Carlson J, Lennvall T, Fohler G. Value-based overload handling of aperiodic tasks in offline scheduled real-time systems. In: Work-in-Progress Session of the 13th Euromicro Conf. on Real-Time Systems. IEEE Computer Society Press, 2001.46~49.

[10] Lennvall T, Carlson J, Fohler G. Value-based overload handling in distributed offline scheduled real-time systems. Technical Report, Department of Computer Engineering, Malardalen University, 2001.

[11] Jin H, Wang HA, Wang Q, Dai GZ. An integrated design method of task priority. Journal of Software, 2003,14(3):376~382 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/376.htm

附中文参考文献:

[11] 金宏,王宏安,王强,戴国忠.一种任务优先级的综合设计方法.软件学报,2003,14(3):376~382.http://www.jos.org.cn/1000-9825/14/376.htm