# 一种实时异构系统的集成动态调度算法*

乔 颖, 邹 冰, 方 亭, 王宏安, 戴国忠

(中国科学院 软件研究所 人机交互与智能信息处理实验室,北京 100080)
E-mail: lisa_qiao@sina.com
http://iel.iscas.ac.cn

摘要: 提出了一种实时异构系统的集成动态调度算法.该算法通过一个新的任务分配策略以及软实时任务的服务质量 QoS(quality of service)降级策略,不仅以统一方式完成了对实时异构系统中硬、软实时任务的集成动态调度,而且提高了算法的调度成功率.同时,还进行了大量的模拟研究.这些模拟以传统的近视算法为基准,将其应用在实时异构系统集成动态调度时的调度成功率与新算法进行比较,模拟结果表明,在多种任务参数取值下,新算法的调度成功率均高于传统的近视算法.

关 键 词: 异构系统;硬实时;软实时;动态调度;调度成功率;启发式搜索;回溯
中图法分类号: TP393 文献标识码: A

随着实时应用范围的加大和复杂度的提高,实时系统中的非周期任务往往具有从硬截止期到软截止期范围的多种时间约束[1],因此,集成调度算法显得十分重要.另一方面,随着近几年来异构计算的兴起,实时异构系统也被广泛应用在航空航天、工业控制、电信行业、图像处理以及 Internet 等诸多领域.

然而,目前所提出的算法却都是用来解决实时同构系统的集成动态调度问题的.例如,文献[2]提出的算法利用了多媒体服务器的方法,并引入 QoS 降级机制,解决了实时多处理系统中多媒体任务和硬实时任务的集成动态调度.文献[3]的算法通过降级软实时任务的结果质量,提高了整体的可调度性.本文对实时异构系统硬、软实时任务的集成动态调度进行了研究,提出了一种新的实时异构系统的集成动态调度算法.

## 1 系统模型

### 1.1 调度器模型

假设实时异构系统有 $m$ 个处理器($m$>1),且这些处理器具有不同的处理速度.调度器采用的是集中式的结构,系统中有一个处理器作为专门的调度器,所有任务都要先到达这个中心调度器,然后被分配到系统中其他的处理器去执行.每个处理器都有自己的一个调度队列,这样,在它执行完当前任务以后,就从其调度队列中取出一个任务来执行.调度器与各处理器之间的通信通过这些调度队列来实现.同时,调度器与各处理器并行地运行,它对新到达的任务进行调度,并周期性地对调度队列进行修改.

### 1.2 实时异构系统的统一任务模型

非精确计算[4]是一个通过妥协结果质量满足任务截止期,并以此来提供调度灵活性的典型模型.该模型尽

* 收稿日期: 2001-03-09; 修改日期: 2001-12-05
基金项目: 国家自然科学基金资助项目(69896250;79931000)
作者简介: 乔颖(1973－),女,北京人,博士,主要研究领域为数据库技术,信息系统;邹冰(1973－),女,浙江温州人,硕士生,主要研究领域为实时智能,CSCW;方亭(1978－),女,浙江人,硕士生,主要研究领域为实时智能,GIS;王宏安(1963－),男,安徽和县人,博士,研究员,主要研究领域为实时智能,多通道用户界面;戴国忠(1944－),男,江苏无锡人,研究员,博士生导师,主要研究领域为多通道用户界面,实时智能,CIMS.

量避免截止期错过现象的发生,并在系统无法及时产生准确结果的时候,通过向用户提供具有可接受质量的近似结果来进行优雅的降级.由于软实时任务的截止期即使被错过也不会引起十分严重的后果,因此,允许其在截止期之前只被部分地完成.于是,我们便可以利用非精确计算模型来构造实时异构系统的统一任务模型.在这个任务模型中,每一个软实时任务都有多个逻辑版本.同一任务的不同逻辑版本具有不同的运行时间.同时,任务的每一个逻辑版本都代表了任务的一个服务等级.

注意到实时异构系统中处理器间的速度差异,我们可以对统一任务模型进行如下描述:

定义 **1**. $Sta(T)$表示任务 $T$ 的可开始时间.

定义 **2**. $Attri(T)$表示任务 $T$ 的属性.

$$Attri(T)=\begin{cases}1, & \text{若任务}T\text{为硬实时任务}\\ 0, & \text{若任务}T\text{为软实时任务}\end{cases}.$$

设系统中有 $m$ 个处理器($m>1$),记为 $p_1,p_2,\ldots,p_m$.$\Gamma$ 表示任务集.于是,实时异构系统硬、软实时任务的统一任务模型可以表示为如下形式:

(1) $\forall T,T\in\Gamma$,均可表示成一个多元组$\theta_T$,且

$$\theta_T=(a_T,r_T,D_T,v_T,ET_T).$$

$a_T$:任务 $T$ 的到达时间,$a_T\geq 0$.$r_T$:任务 $T$ 就绪时间,$r_T\geq 0$,且 $r_T\geq a_T$.$D_T$:任务 $T$ 的截止期,$D_T\geq 0$.$v_T$:任务 $T$ 所具有的不同逻辑版本的数目,同时也表示了任务 $T$ 的服务等级数,

$$v_T\begin{cases}=1, & Attri(T)=1\\ >1, & Attri(T)=0\end{cases},$$

$ET_T$:任务 $T$ 的运行时间.对于硬实时任务,$ET_T$ 为一个向量;对于软实时任务,$ET_T$ 则为一个矩阵.

$$\forall T,T\in\{T\mid Attri(T)=1\},\text{均有}ET_T=(et(T)_{p_1},et(T)_{p_2},...,et(T)_{p_m}),$$

其中 $et(T)_{p_j}$ 表示任务 $T$ 在处理器 $p_j(j=1,\ldots,m)$上运行时的最坏运行时间,且$\forall j,j=1,\ldots,m$,均有 $r_T+et(T)_{p_j}\leq D_T$.

$$\forall T,T\in\{T\mid Atrri(T)=0\},\text{均有}ET_T=(et(T)_{ip_j})_{v_T\times m},i=1,...,v_T;\ j=1,...,m,$$

其中 $et(T)_{ip_j}$ 表示任务 $T$ 的逻辑版本 $i$(即服务等级 $i$)在处理器 $p_j(j=1,\ldots,m)$上运行时的最坏运行时间.同时,$\forall j,1\leq j\leq m$,都有 $et(T)_{1p_j}\geq et(T)_{2p_j}\geq\ldots\geq et(T)_{v_Tp_j}$,且$\forall i,\forall j,1\leq i\leq v_T,1\leq j\leq m$,均有 $r_T+et(T)_{ip_j}\leq D_T$.

(2) 任务是不可抢占的,且相互间是独立的.

(3) 任务不具有并行性.

(4) 除了处理器以外,任务还可能需要其他一些资源,如某些数据结构、变量及缓冲区等.每个任务对资源的访问方式包括互斥访问和共享访问两种.

## 2 新的实时异构系统的集成动态调度算法

### 2.1 相关定义

定义 **3**. 在一个调度中,若某任务的时间约束和资源需求都可以满足,则称该任务在这个调度中是可行的,任务的可行性则是指任务在调度中是“可行的”的机率.同时,如果一个任务集中的所有任务在调度中都是可行的,则称该调度对于此任务集来说是一个可行调度[5,6].

定义 **4**. 局部调度是指任务子集的一个可行调度.若一个局部调度被任务集中所剩的任何一个任务扩展后仍是可行调度,则称该局部调度是强可行调度[6].

定义 **5**. $EAT_k^s$ 是资源 $R_k$ 在共享访问方式下的最早可用时间;$EAT_k^e$ 是资源 $R_k$ 在互斥访问方式下的最早可用时间.

定义 **6**. $ESTP(T,P)$表示任务 $T$ 在处理器 $P$ 上的最早可开始时间.假设 $R_T$ 为任务 $T$ 所需的资源集合.于是有

$$ESTP(T,P)=\mathrm{Max}(r_T,availtime(P),\mathrm{Max}_{R_k\in R_T}EAT_k^u),$$

其中 $availtime(P)$表示处理器 $P$ 的最早可用时间,$\mathrm{Max}_{R_k\in R_T}EAT_k^u$ 表示任务 $T$ 所需资源的最早可用时间,且当资源为

共享访问方式时,$u=s$.当资源为互斥访问方式时,$u=e$.

定义 **7**. $IEST(T)$为表示任务 $T$ 的理想最早可开始时间.假设 $PE$ 为一个处理器集合.于是有

$$IEST(T)=\mathrm{Min}_{P\in PE}ESTP(T,P).$$

定义 **8**. $avail(T,P)$表示任务 $T$ 在处理器 $P$ 上的可行性,即任务 $T$ 在处理器 $P$ 上运行是否能满足其截止期.其中,若 $Attri(T)=1$,则有

$$avail(T,P)=\begin{cases}1, & ESTP(T,P)+et(T)_p\le D_T\\ 0, & ESTP(T,P)+et(T)_p>D_T\end{cases}.$$

若 $Attri(T)=0$,则有

$$avail(T,P)=\begin{cases}1, & \exists i,1\le i\le v_T, ESTP(T,P)+et(T)_{ip}\le D_T\\ 0, & \forall i,1\le i\le v_T, ESTP(T,P)+et(T)_{ip}>D_T\end{cases}.$$

定义 **9**. $sysavailtime$ 表示系统中所有处理器最早可用时间的最小值,即 $sysavailtime=\mathrm{Min}_{P\in PE}(availtime(P))$.

定义 **10**. $Spe(P)$反映了处理器 $P$ 的速度.处理器 $P$ 的速度越慢, $Spe(P)$的值越大.

定义 **11**. $Finishtime(T,P)$表示任务 $T$ 在处理器 $P$ 上的完成时间,

$$Finishtime(T,P)=\begin{cases}ESTP(T,P)+et(T)_P, & Attri(T)=1\\ ESTP(T,P)+et(T)_{iP}, & Attri(T)=0\end{cases},$$

其中 $i$ 表示任务当前所处的任务等级,$i=1,\ldots,v_T$.

### 2.2 任务分配策略

为了处理实时异构系统中的动态调度问题,我们在启发式搜索[5]中引入了一个新的任务分配策略.该策略充分考虑了实时异构系统的异构性,并通过引入一个目标函数 $S$ 作为任务分配时选择处理器的依据,提高了未被调度任务的可行性,从而达到了提高算法调度成功率的目的.

从启发式搜索的角度来看,如果未被调度任务的可行性较高,则当前调度通过可行性检查被确认为强可行调度的机率就会比较大[7,8].这样就可以减少回溯的发生,从而使整个算法的调度成功率增大.因此,在进行任务分配时,我们应尽量选择那些能使运行完任务 $T$ 后未被调度任务可行性较高的处理器.

显然,速度快和最早可用时间早的处理器具有更大的可能性来满足任务的截止期,因此,在运行完任务 $T$ 后,若 $sysavailtime$(系统的最早可用时间)越早且最早可用时间为 $sysavailtime$ 的处理器的速度越快,则未被调度任务的可行性越大.于是,我们为每一个处理器定义了一个新的目标函数 $S$.对于每一个处理器 $P$,目标函数 $S$ 可表示为

$$S(P)=availtime(P)+W_p*Spe(P),$$

其中 $W_p$ 表示权值,$W_p>0$.

新的任务分配策略的基本思想是,使用目标函数 $S$ 为任务选择处理器去运行,并总是在可以满足任务 $T$ 截止期的处理器中选择目标函数 $S$ 值最大的来运行任务 $T$.这样,便在为任务 $T$ 选择处理器时折衷地考虑了每个处理器的最早可用时间和速度,并在保证任务 $T$ 截止期的情况下,尽量将任务 $T$ 分配到速度较慢且最早可用时间较晚的处理器上,而将那些速度快和最早可用时间早的处理器尽可能地保留下来,从而通过加大未被调度任务截止期被满足的概率提高了未被调度任务的可行性.

此外,考虑到在有些情况下任务还需要除处理器之外的资源,因此,在新的任务分配策略中,我们要根据任务 $T$ 使用资源的情况来判断是否使用目标函数 $S$ 来为任务 $T$ 分配处理器.判断依据是,如果采用了上述的目标函数方法,任务 $T$ 所占资源在互斥访问下的最早可用时间被推迟后,不会影响到未被调度任务截止期的满足情况,则可以采用该方法为任务 $T$ 选择处理器去运行;否则,我们将在满足任务 $T$ 截止期的处理器中为其选择一个最早可完成它的处理器.

综上所述,我们可以将新的任务分配策略命名为函数 $choosep(T)$,该函数返回一个被选中用来运行任务 $T$ 的处理器的编号.

设 $ESTR_T$ 为任务 $T$ 所需资源在互斥访问下的最早可用时间,$\Gamma_{us}$ 表示未被调度的任务集,$Res(T)$为任务 $T$ 所

需的资源集合,mod(*T*,*R*)表示任务 *T* 对资源 *R* 的访问方式,且有

$$\mathrm{mod}(T,R)=\begin{cases}s, & T\text{以共享方式访问}R\\ e, & T\text{以互斥方式访问}R\end{cases}.$$

于是,新的任务分配策略可表示如下:

(1) 若 $Res(T)=\varnothing$,则为任务 *T* 选择一个处理器 *P*,使得

$$S(P)=\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}S(pe).$$

(2) 若 $Res(T)\neq\varnothing$,即任务 *T* 除处理器以外,还需要其他资源,我们采用以下方法:

(2.1) 检索任务 *T* 的资源使用情况.

(2.2) 若对于$\forall T_{us},T_{us}\in\Gamma_{us}$,都有 $Res(T_{us})\cap Res(T)=\varnothing$,则任务 *T* 对处理器的选择方法与(1)相同;

(2.3) 若对于$\forall R,R\in Res(T)$,均有 $\mathrm{mod}(T,R)=s$,且对于$\forall T_{us},T_{us}\in\Gamma_{us},\forall RT,RT\in Res(T)$,均有 $\mathrm{mod}(T_{us},RT)=s$,则任务 *T* 对处理器的选择方法与(1)相同;

(2.4) 其他情况下,分以下几种情况来处理:设处理器 *P*′满足以下约束,即

$$availtime(P')=\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}availtime(pe).$$

(2.4.1) 若 $r_T\le ESTR_T\wedge ESTR_T=\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}availtime(pe)\wedge Spe(P')=\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}Spe(pe)$,则任务 *T* 对处理器的选择方法与(1)相同;

(2.4.2) 若 $r_T\ge ESTR_T\wedge r_T\ge\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}availtime(pe)\wedge Spe(P')=\mathrm{Max}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1\}}Spe(pe)$,则任务 *T* 对处理器的选择方法与(1)相同;

(2.4.3) 其他情况下,为 *T* 选择一个处理器 *P*,使得

$$Finishtime(T,P)=\mathrm{Min}_{pe\in\{pe|pe\in PE\wedge avail(T,pe)=1}Finishtime(T,P).$$

### 2.3 软实时任务的服务质量(QoS)降级策略

在新算法中,我们提出了一个软实时任务的服务质量(QoS)降级策略.该策略利用了软实时任务的不精确计算模型,通过在必要时妥协软实时任务的结果质量提高了算法整体的调度成功率,并为以统一形式对实时异构系统中的硬、软实时任务进行集成动态调度提供了有效的方法.

在该服务质量降级策略中,对于可行性检查窗口中的每个软实时任务,一开始,我们都让它处于其所具有的最高服务等级,并使用与该服务等级相对应的运行时间来进行可行性检查.这一做法的目的是为了在保证任务截止期的前提下,使被降级任务尽可能地获得最好的结果质量(服务质量).

这样,在启发式搜索中,一旦发现由于检查窗口中某个软实时任务的不可调度性而导致当前调度不是强可行时,该软实时任务的服务等级便会被降低一级,即为任务 *T* 选择一个运行时间仅次于当前服务等级的逻辑版本.这种降级会一直持续,直到该任务通过可行性检查为止.同时,当该软实时任务的服务等级已降至最低,而可行性检查仍没有成功时,可能会引发一次回溯.

我们将这个降级策略记为函数 *degrade*(*T*).其中 *T* 表示任务,而函数的返回值则为任务 *T* 被允许的实际服务等级.函数 *degrade*(*T*)的细节如下所示:

(1) If 导致当前调度为非强可行调度的任务为软实时任务 then

(1.1) 检查任务 *T* 的当前服务等级;

(1.2) If 任务 *T* 的当前服务等级不为最低的一级 then

将任务 *T* 的服务等级降级一等,即为任务 *T* 选择一个运行时间仅少于当前服务等级的逻辑版本;

(1.3) 重复(1.1)和(1.2),直到任务 *T* 的服务等级已降至最低,或者可行性检查获得成功;

(2) else exit.

### 2.4 算法描述

新的实时异构系统中的硬、软实时任务的集成动态调度算法如下所示:

设 *K* 为可行性检查窗口的大小,$d_T$ 为任务 *T* 的截止期,*W* 为 *H* 函数的权值.

(1) 将任务队列中的任务按截止期的非递减顺序排列.开始时,局部调度为空.

(2) 检查可行性检查窗口中的 $K$ 个(或少于 $K$ 个)任务.

(3) For $i$=1 to $K$ (or less than $K$)

(3.1) If 使用任务 $T_i$ 对当前调度进行扩充后为不可行的

Then $degrade(T_i)$;

(4) 通过对可行性检查窗口中的 $K$ 个(或少于 $K$ 个)任务进行可行性检查,决定当前的局部调度是否为强可行的.若是,则 feasible=true;否则,feasible=false;

(5) If (feasible==true)

(5.1) 计算可行性检查窗口中的任务的目标函数 $H$ 值,其中

$$H(T)=d_T+W*IEST(T);$$

(5.2) 选择目标函数 $H$ 值最小的任务 $T$ 扩充当前调度,并调用 $choosep(T)$为任务 $T$ 分配处理器;

Else

(5.3) 回溯到上一层的调度;

(5.4) 在此层的可行性检查窗口中,选择目标函数 $H$ 值次优的任务 $T'$扩充当前调度,并调用 $choosep(T')$为任务 $T'$分配处理器;

(6) 将可行性检查窗口向后移动一个任务;

(7) 重复(2)~(6)的操作,直到以下条件中的任意一个得到满足:

(a) 找到了一个完全的可行调度;

(b) 已达到了最大的回溯次数或 $H$ 函数的最大估算值;

(c) 已经没有再回溯的可能.

## 3 模拟研究及结果

为了评估新算法的性能,我们进行了一系列的模拟研究.由于调度成功率[5]是实时调度算法最重要的性能评价标准,因此,在这些模拟研究中,我们仅考虑算法的调度成功率.

### 3.1 任务生成方法

定义 **12**. $Rand(x,y)$表示 $x$ 与 $y$ 之间的随机数.

生成实时异构系统中,硬、软实时任务相混合的可调度任务集的方法如下:

(1) 设系统中有 $m$ 个处理器,记为 $p_1,p_2,\ldots,p_m$,从中选取一个速度最慢的处理器作为基准处理器.为方便起见,假设其他各处理器的速度分别为这个基准处理器速度的不同倍数.我们将这些倍数记为一个速度倍数向量 $\lambda$,$\lambda=(\lambda_1,\ldots,\lambda_m)$,$\lambda_1\leq\ldots\leq\lambda_m$,且有

$$\lambda_{i+1}/\lambda_i=\lambda_i/\lambda_{i-1},i=2,...,m-1.$$

(2) 令 $\beta$ 为速度增量,使得 $\beta=\lambda_{i+1}/\lambda_1,i=1,\ldots,m-1$.这样,$\beta$ 便反映了各处理器间的速度差异.$\beta$ 值越大,则处理器间的速度差异越大.

(3) 任务集中的任务要在各个处理器上不断地生成,一直达到调度长度为止,同时,在各个处理器上不能有空闲时间[5].

(4) 对于每一个被生成的任务 $T$,其为软实时任务的概率是 $Task\_P$,为硬实时任务的概率是 $1-Task\_P$.

(5) 任务 $T$ 逻辑版本(服务等级)数 $v_T$ 的取值方法为(Max_$v$ 为软实时任务的最大逻辑版本数):

$$v_T=\begin{cases}1, & T\text{为硬实时任务}\\ Rand(1,\mathrm{Max}\_v), & T\text{为软实时任务}\end{cases}.$$

(6) 若任务 $T$ 为硬实时任务,则其运行时间的取值方法为(Min_$C$ 和 Max_$C$ 分别为任务的最小和最大运行时间).

$$et(T)_{p_k}=Rand(\mathrm{Min}\_C,\mathrm{Max}\_C),$$

其中 $p_k$ 为基准处理器.这样,根据速度倍数向量 $\lambda$,便可从 $et(T)_{p_k}$ 分别计算出任务 $T$ 在其他 $m-1$ 个处理器上的运行时

间 $et(T)_{p_j}$,$j$=1,…,$k$–1,$k$+1,…,$m$.

(7) 若任务 $T$ 为软实时任务,则其运行时间的取值方法为

$$et(T)_{ip_k} = Rand(\text{Min}\_C, \text{Max}\_C),$$

且

$$et(T)_{sp_k} < et(T)_{(s+1)p_k}, s = 1,...,v_T - 1,$$

其中 $p_k$ 为基准处理器.这样,根据速度倍数向量$\lambda$,可从 $et(T)_{ip_k}$ 分别计算出任务 $T$ 处于各服务等级 $i$ 时,在其他 $m$–1 个处理器上的运行时间 $et(T)_{ip_j}$($i$=1,…,$v_T$;$j$=1,…,$k$–1,$k$+1,…,$m$),且 $et(T)_{sp_j} < et(T)_{(s+1)p_j}$ ($s$=1,…,$v_T$–1).

(8) 任务截止期 $D_T$ 的取值方法为

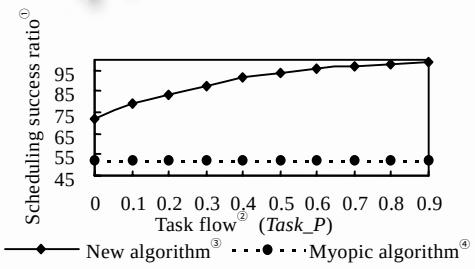
$$D_T = Rand(SC, (1+R)*SC),$$

其中 $SC$ 表示任务集中最晚完成的任务的结束时间,$R$ 为任务的可延迟度.

(9) 对于每一个资源,根据 *Use_P* 和 *Share_P* 的取值来确定任务 $T$ 对该资源的访问情况.即任务以 *Use_P* 的概率访问该资源,以 *Share_P* 的概率对该资源进行共享式访问.

### 3.2 模拟结果及分析

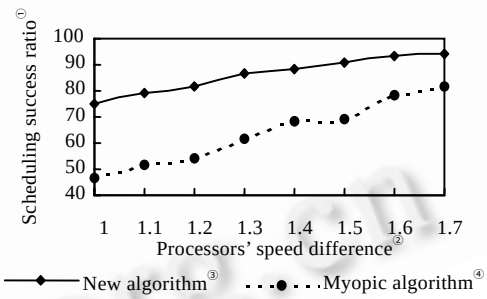
在模拟中,我们采用上述方法产生了 200 个可调度任务集,固定地将调度长度设置为 800.每个任务集包含了 30~60 个任务.同时,将系统中的处理器个数和资源个数均设为 3,并假设每个资源只有一个实例.此外,算法所允许的最大回溯次数、可行性检查窗口大小 $K$,$H$ 函数的权值 $W$,$S$ 函数的权值 $W_P$,*Share_P*,Max_$v$ 的值被分别设置为 1,7,2,2,0.5 和 10.我们将近视算法应用到了实时异构系统的集成动态调度上,并以此作为基准,在多种任务参数的取值下,与新算法的调度成功率进行了比较.模拟结果如图 1~图 4 所示.图中每一个点的数据都是重复运行了 5 次以后的平均结果.




①调度成功率,②任务流,③新算法,④近视算法.

Fig.1 Effect of task flow (*Task_P*) on scheduling success ratio

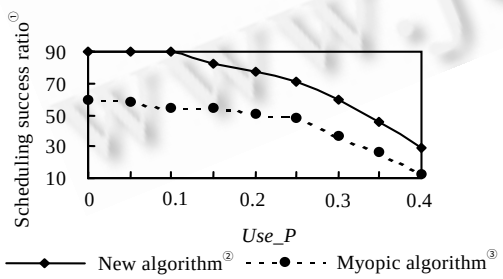
图 1 任务流(*Task_P*)对调度成功率的影响




①调度成功率,②速度差异,③新算法,④近视算法.

Fig.2 Effect of processors' speed difference on scheduling success ratio

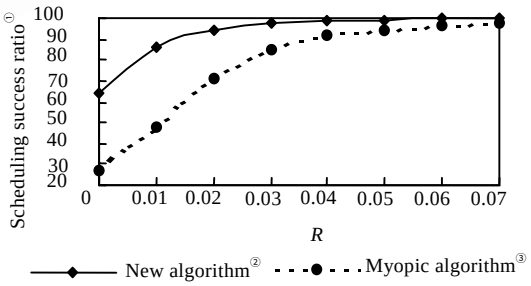
图 2 处理器速度差异($\beta$)对调度成功率的影响




①调度成功率,②新算法,③近视算法.

Fig.3 Effect of resource usage (*Use_P*) on scheduling success ratio

图 3 资源使用情况(*Use_P*)对调度成功率的影响




①调度成功率,②新算法,③近视算法.

Fig.4 Effect of laxity ($R$) on scheduling success ratio

图 4 任务的可延迟度($R$)对调度成功率的影响

3.2.1 任务负载(*Task_P*)对调度成功率的影响

任务负载(*Task_P*)对调度成功率的影响如图 1 所示.在这个模拟过程中,我们将 *Use_P*,$\beta$和 *R* 的值分别设定为 0.5,1.1 和 0.01.模拟结果显示,随着 *Task_P* 的增加,即增加软实时任务在任务集中的比例,新算法的调度成功率将会增加,而任务流的变化对近视算法的调度成功率则没有影响.这是因为,随着软实时任务在任务集中比例的增加,新算法将有更多的机会在可行性检查失败时对软实时任务进行服务质量的降级,这显然会导致算法调度成功率的增加.对于近视算法来说,它对软实时任务的调度与硬实时任务完全相同,因此,无论任务流如何变化,其调度成功率都不发生变化.

同时,图 1 还显示,新算法的调度成功率在任务流发生变化的过程中始终高于传统的近视算法.这一现象是由两个原因引起的:第 1,在必要时,新算法对软实时任务的服务等级进行了降级,从而通过妥协软实时任务的服务质量加大了任务的可行性,进而提高了算法整体的调度成功率;第 2,在新算法中提出了一个新的任务分配策略,该策略充分考虑了实时异构系统的异构性对调度决策的影响,通过在为任务分配处理器时折衷考虑处理器的最早可用时间和速度,提高了未被调度任务的可行性,显然,这也使算法的调度成功率得到了提高.

3.2.2 处理器之间的速度差异($\beta$)对调度成功率的影响

图 2 反映了处理器间的速度差异对调度成功率的影响.在这个模拟过程中,*Task_P*,*Use_P* 和 *R* 的值分别被设置为 0.1,0.2 和 0.01.从模拟结果可以看出,由于$\beta$值的增大,处理器的速度普遍提高,因此,随着$\beta$值的增大,两个算法的调度成功率都将随之增加,而且近视算法对速度差异的敏感度要大于新算法.同时,由于新算法中引入了新的任务分配策略和软实时任务服务质量降级策略,因此,在$\beta$的变化过程中,新算法的调度成功率始终高于近视算法.尤其是在 $1\leq\beta<1.5$ 时,新算法在调度成功率上的优势更为明显,平均要比近视算法高出 20%~30%.

3.2.3 任务的资源使用情况(*Use_P*)对调度成功率的影响

图 3 显示了当任务使用资源情况(*Use_P*)发生变化时,两个算法调度成功率的变化情况.在这一模拟过程中,我们将 *Task_P*,$\beta$和 *R* 分别取值为 0.1,1.1,0.01.模拟结果显示,*Use_P* 的增大将导致两个算法调度成功率的下降,同时,新算法和近视算法对任务使用资源的概率都比较敏感,且相比之下,新算法的敏感度更强.此外,在 *Use_P* 的整个变化过程中,新算法的调度成功率要比近视算法平均高出 26%.

3.2.4 任务可延迟度(*R*)对调度成功率的影响

图 4 表现了任务的可延迟度对调度成功率的影响.在模拟中,我们将$\beta$,*Use_P* 和 *Task_P* 的值分别设定为 1.1,0.2 和 0.1.模拟结果显示,当任务的可延迟度增加时,两个算法的调度成功率也随之增加,且近视算法对任务可延迟度的敏感度要高于新算法,在 *R* 值的整个变化过程中,近视算法调度成功率的变化幅度达到了 70%,新算法则只变化了 35%.此外,从图 4 中还可以看出,在 *R* 值的变化过程中,新算法的调度成功率始终高于近视算法,且其优势随着任务可延迟度的增加而逐渐减小.这一现象表明,当接受调度的任务集中任务的截止期分布比较密集时,即任务间对 CPU 时间的竞争比较激烈时,采用新算法进行调度的效果较好.

## 4 结 论

许多动态非周期任务都具有多种时间约束,这些约束涉及到从硬截止期到软截止期的范围.然而,目前还没有提出来解决实时异构系统中硬、软实时任务的集成动态调度问题的有效方法.本文对这一问题进行了深入研究,提出了一种实时异构系统的集成动态调度算法.在这个算法中,我们提出了一个新的任务分配策略和一个软实时任务服务质量(QoS)降级策略.在任务分配策略中,我们引入了目标函数 *S*,并利用这个目标函数为任务选择最合适的处理器来运行,从而达到了提高算法调度成功率的目的.同时,软实时任务的 QoS 降级策略则通过在必要时妥协软实时任务的结果质量来提高算法整体的调度成功率.此外,为了评估新算法的性能,本文还进行了一系列的模拟研究.在这些模拟中,我们将近视算法应用在实时异构系统的集成动态调度中,并以此为基准,与新算法的调度成功率进行了比较.模拟结果显示,在多种任务参数的取值下,新算法的调度成功率均优于近视算法,且其优势平均在 30%左右.

**References:**

[1] Lehoczky, J.P., Ramos-Thuel, S. An optimal algorithm for scheduling soft-aperiodic tasks in fixed priority preemptive systems. In: Proceedings of the 13th IEEE Real-Time System Symposium. IEEE Computer Society Press, 1992. 110 ~123.

[2] Kaneko, H., Stankovic, J.A., Sen, S., *et al*. Integrated scheduling of multimedia and hard real-time tasks. In: Proceedings of the 17th IEEE Real-Time System Symposium. IEEE Computer society Press 1996. 206~219.

[3] Manimaran, G., Murthy, C.S.R. Integrated dynamic scheduling of hard and QoS degradable real-time tasks in multiprocessor systems. In: Proceedings of the 5th International Conference on Real-time Computing Systems and Applications. Japan, 1998.

[4] Liu, J.W.S, Shin, W.K., Liu, K.J., *et al*. Imprecise computations. Proceedings of the IEEE, 1994, 82(1):83~94.

[5] Ramamritham, K., Stankovic, J.A., Shiah, P.-F. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems, 1990,11(2):184~194.

[6] Manimaran, G., Murthy, C.S.R. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Transactions on Parallel and Distributed Systems, 1998,19(3):312~319.

[7] Qiao, Ying, Wang, Hong-an, Dai, Guo-zhong. Developing a new dynamic scheduling algorithm for real-time multiprocessor systems. Journal of Software, 2002,13(1):51~58 (in Chinese).

[8] Qiao, Ying, Zou, Bing, Wang, Hong-an, *et al*. Developing a new dynamic scheduling algorithm for real-time heterogeneous systems. In: Lin, Yen-Chun, Shen, Hong, eds., Proceedings of the 2nd International Conference on Parallel and Distributed Computing, Applications, and Technologies. 2001. 8~15.

[9] Tan, S.T., Wang, T.N., Zhao, Y.F., *et al*. A constrained finite element method for modeling cloth deformation. Visual Computer, 1999,15(2):90~99.

附中文参考文献:

[7] 乔颖,王宏安,戴国忠.一种新的实时多处理器系统的动态调度算法.软件学报,2002,13(1):51~58.

# Design and Evaluation of an Algorithm for Integrated Dynamic Scheduling in Real-Time Heterogeneous Systems*

QIAO Ying, ZOU Bing, FANG Ting, WANG Hong-an, DAI Guo-zhong

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

E-mail: lisa_qiao@sina.com

http://iel.iscas.ac.cn

**Abstract:** In this paper, an efficient algorithm is presented to dynamically schedule the task sets combining hard and soft real-time tasks in heterogeneous systems. The proposed algorithm improves the scheduling success ratio by introducing a new task assignment policy and a QoS (quality of service) degradation policy for soft real-time tasks. To evaluate the performance of the new algorithm, extensive simulation studies have been done. These simulations apply myopic algorithm to schedule the hard and soft real-time tasks in heterogeneous systems and use it as a baseline to compare with the proposed algorithm. Simulation results show that the scheduling success ratio of the new algorithm is always higher than that of myopic algorithm in real-time heterogeneous systems for a variety of task parameters.

**Key words:** heterogeneous system; hard real-time; soft real-time; dynamic scheduling; scheduling success ratio; heuristic; backtrack

* Received March 9, 2001; accepted December 5, 2001

Supported by the National Natural Science Foundation of China under Grant Nos.69896250, 79931000