弱硬实时系统任意窗口约束调度研究

吴 彤[1], 金士尧[1], 陈积明[2]
(1. 国防科学技术大学并行与分布处理国家重点实验室, 湖南长沙 410073;
2. 浙江大学工业控制技术国家重点实验室, 浙江杭州 310027)

摘 要: 弱硬实时应用中的 QoS 在过载情况下会不同程度地退化. 本文针对因仅考虑无限区间或者某一固定有限区间上的任务丢失率而导致重度过载情况下服务不公平的情况, 基于可变区间, 提出 (p, k) 约束, 给出任意窗口约束调度(Any Window Constraint Schedule, AWCS)算法及其简化算法 K 窗口约束调度(K-Window Constraint Schedule, KWCS), 实验表明 KWCS 与 AWCS 的性能相当, 且开销大幅降低. 通过分析算法特性, 给出具有 QoS 保证的时延上界通用表示方法. 实验结果表明在重度过载情况下, AWCS(KWCS)优于其它弱硬实时算法.

关键词: 任意窗口约束调度; 弱硬实时; K 窗口约束调度; 区间最小成功率; 时延
中图分类号: TP316 文献标识码: A 文章编号: 0372-2112 (2008) 08-1564-07

Research on Any Window Constraint Scheduling in Weakly Hard Real-Time System

WU Tong[1], JIN Shi-yao[1], CHEN Ji-ming[2]
(1. *National Laboratory of Parallel and Distributed Processing, NUDT, Changsha, Hunan 410073, China;*
2. *State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou, Zhejiang 310027, China*)

Abstract: In overload conditions, the QoS of applications with weakly hard real-time constraint is degraded diversely. To deal with the unfairness case caused by only considering an infinite interval or a fixed finite window loss-rate, this paper brings up a concept with a (p, k) constraint, which is based on variable interval. Based on the (p, k) constraint, an algorithm is proposed, named as AWCS(Any Window Constraint Scheduling). A simple version of AWCS is put forward, which is called KWCS(K-Window Constraint Schedule). Extensive experiments show that KWCS can supersede AWCS, and not only achieve comparative performance but also get lower complexity. The properties of two algorithms are addressed, and a general representation of delay bound of the scheduling algorithms is brought forward. Results show that both AWCS and KWCS can provide better performance than other weakly hard real-time schedule algorithms in heavily overload circumstances.

Key words: any window constraint schedule; weakly hard real-time; K-window constraint schedule; minimum success ratio of an interval; delay

1 引言

随着计算机、网络技术的飞速发展, 出现了许多基于网络的实时应用, 例如网络多媒体等, 这类应用称为实时网络应用. 实时网络应用多属于弱硬实时的范畴, 要求提供一种具有不同可靠性、实时性和稳定性的服务质量(Quality of Service, QoS). 所谓 QoS 是指"在网络中提供资源保证和服务区分的能力"[1], 一般包括对时延、抖动和丢失率三个参数的评价[2].

许多研究者从丢失率出发定义实时应用的 QoS. 在网络应用中, 特别是在网络多媒体应用中, 允许某些任务错过截止期, 但是不同用户对截止期错过的容忍程度不同. Chou 等[3]提出统计实时通道, 从概率的角度给出了三种 QoS 保证: (1) $P(\text{delay} \leqslant D) \geqslant Z$, 任务实例的延时不大于截止期的概率不小于某一给定值 Z, 考虑的是无限或充分长区间内的丢失率, 相应研究包括文献[3,4]中给出的调度算法; (2) $P(\text{no misses in any time interval of a certain length}) \geqslant Z$ 在固定长度窗口内没有截止期丢失的概率不小于 Z, 考虑的是有限区间内的丢失率, 相应算法包括 Skip-over[5]、DBP[6]、Melody 系统[7]、BMS[8]、MAA 与 MRA[9]、DWCS[10]和 VDS[11]; (3) 在任意长度大于某一特定长度的窗口内, 都满足 $P(\text{delay} \leqslant D) \geqslant Z$, 这里

收稿日期: 2007-06-20; 修回日期: 2008-04-16
基金项目: 国家自然科学基金(No. 60603032, 60604029); NSFC-广东省联合基金(No. U0735003)

考虑了可变长度区间的丢失率, 相关研究包括陈积明等提出的$(\overline{m}, p)$[12]和 CDBS[13].

由于网络的不确定性, 有时会出现任务流突发的情况, 即在短时间内出现过载甚至是严重过载, 预定义的 QoS 会受到破坏, QoS 会有不同程度的退化, 因此研究在高负载下 QoS 性能的缓慢退化具有重要意义.

经典弱硬实时调度算法在高负载下性能会受到较大影响而带来不公平性, 例如 DBP[6] 会出现饿死现象; 尽管 DWCS[10] 控制了延时边界, 但是对于具有同样 QoS 需求的任务, 其实际服务质量相差很大; HAS 启发式算法(Heuristic Algorithm Sckedule, HAS)[4] 虽然考虑了丢失截止期的分布, 但是实验显示, HAS 对大周期任务的服务质量高于小周期任务. 这种不公平性是系统所不希望的. 因此不论系统是否过载, 对于具有相同 QoS 需求的任务, 需要提供相近的 QoS.

究其原因, 现有的算法在过载情况下的不公平性是由于 QoS 保证的粒度问题而引起的, 第一类保证仅考虑整体则粒度过大; 而第二类保证只考虑有限区间, 在过载情况下会忽略历史任务的累积影响; 只有考虑可变区间才能更好地平衡 QoS 的退化, 使其在各类任务之间公平地、缓慢地退化.

针对上述问题, 本文基于可变区间, 提出(p, k)约束, 并针对(p, k)约束, 设计了 AWCS 算法, 由于 AWCS 在重度负载情况下复杂度不收敛, 提出简化算法 KWCS, 实验表明 AWCS 和 KWCS 性能出众.

2　相关背景

2.1　(p, k)约束

Bernat 提出的四类弱硬实时约束[14]都是针对固定区间上满足截止期的情况, 没有综合考虑任务连续丢失和整体任务丢失率, 因此, 陈积明等根据连续丢失次数和满足比例定义了$(\overline{m}, p)$约束[12]. $(\overline{m}, p)$隐含定义了最小窗口长度 $\left\lceil \frac{n}{\lfloor \frac{n}{\overline{m}} \rfloor} \frac{m}{1-p} \right\rceil$, 实际上当最小窗口长度确定后, 也确定了最大连续丢失截止期的次数, 因此为了突出最小窗口长度, 本文使用(p, k)约束描述 QoS 需求, 即要求在任意长度不小于 k 的窗口内, 成功率不小于 p. 显然当(p, k)约束中的 k 值与$(\overline{m}, p)$约束隐含的最小窗口长度相等时, 两种约束等价.

$$(\overline{m}, p) \Leftrightarrow (p, k) \ \text{iff}\ k = \left\lceil \frac{n}{\lfloor \frac{n}{\overline{m}} \rfloor} \frac{m}{1-p} \right\rceil$$

(p, k)约束属于第三类 QoS 保证[3]. 基于(p, k)约束, 在可变窗口下考虑任务的执行情况, 能够获得精确的执行情况分布, 进而可以保证任务的执行序列中所有长度大于 k 的窗口的最小成功率与目标成功率之间的差距尽量小, 这意味着在过载时各类任务的 QoS 退化的程度接近, 提供的服务更加公平.

2.2　相关定义和定理

定义 1　在二进制序列中, 由 1 变为 0 时 0 的位置称为转折点.

定义 2　指定 Σ 为集合$\{0, 1\}$中的元素, Σ^+ 为长度大于 0 的二进制序列集. Σ^n 则是长度为 n 的二进制序列集. 对于任意 $\omega \in \Sigma^+$, 该序列的长度 $l(\omega) = |\omega|$. ω 中 1 的个数表示为 $l^1(\omega)$, 0 的个数表示为 $l^0(\omega)$.

引理 1　对于任意 $\omega \in \Sigma^+$, 将其进行划分, 使得前面 k 位组成序列 ω', 其余位组成序列 ω'', 且 $\frac{l^1(\omega')}{l(\omega')} \geqslant \frac{l^1(\omega'')}{l(\omega'')} \geqslant p$, 那么对于任意序列 ω''', 如 $\frac{l^1(\omega''\omega''')}{l(\omega''\omega''')} \geqslant p$, 必有 $\frac{l^1(\omega\omega'')}{l(\omega\omega'')} \geqslant p$, 如果 $\frac{l^1(\omega\omega''')}{l(\omega\omega''')} < p$, 必然有 $\frac{l^1(\omega''\omega''')}{l(\omega''\omega''')} < p$.

引理 2　如果存在违背约束的子序列, 那么必然存在一个首位为 0 的违背约束的子序列, 或者存在一个长度等于基准窗口长度的违背约束的子序列.

定理 1　根据序列测试和裁剪算法, 如果任务 τ_i 的执行序列存在违背约束$(\overline{m}_i, p_i)$的子序列, 那么在裁剪后的执行序列中, 也必存在违背约束$(\overline{m}_i, p_i)$的子序列.

引理 1、引理 2 和定理 1 的证明见文献[13].

3　AWCS

3.1　算法思想

基于(p, k)约束, 提出窗口长度不小于 k 的调度算法, 称为任意窗口约束调度(Any Window Constraint Schedule, AWCS)算法. 在序列裁剪时, 保存所有转折点和最后 k 个任务实例的执行信息, 通过比较当前最危险子序列中的任务实例数 y 和截止期错过的次数 x 进行调度.

定位最危险子序列方法: 首先根据子序列中的丢失率, 丢失率越大越容易违背约束; 丢失率相等时, 长度越短越容易违背约束. 为说明定位最危险子序列的方法, 证明如下定理.

定理 2　通过判断所有转折点后的序列或最后 k 个任务实例组成的子序列, 可以找到序列中丢失率最大的长度不小于 k 的子序列.

证明　假设序列中存在一个丢失率最大且长度不小于 k 的子序列 ω, 但是 ω 既不是某一个转折点后面的子序列, 也不是最后 k 个任务实例组成的子序列.

删除 ω 中第一个 0 前面的 1, 得到一个首位为 0 的子序列 ω':

- 如果 $l(\omega') \geqslant k$, 那么显然 ω' 的丢失率大于 ω 的丢失率, 且 ω' 是某个转折点后面的子序列, 矛盾;
- 如果 $l(\omega') < k$, 则取 ω 的最后 k 位组成 ω'', 显然

ω''的丢失率大于 ω, 矛盾.

所以假设不成立. 证毕.

获取最危险子序列的方法: 首先根据定理 2 获取长度不小于 k 且丢失率最大的所有子序列, 然后选取其中长度最短的子序列. 如果序列长度小于 k, 在序列前端补充适当数量的成功实例使其长度达到 k. 举例说明如下, 在$(0.6, 5)$约束下, 对于序列 001110111, 最危险子序列是 001110111, 如果下一个任务实例错过截止期, 则下一个任务实例执行后的子序列是 0011101110, 显然这时最危险子序列是01110; 对于另一序列 1100, 由于长度不足 5, 因此在前面添加成功任务实例, 得到这时的最危险子序列 11100.

由引理 1、引理 2 和定理 1 可以保证违背约束的序列不会被裁剪, 因此可以保证 AWCS 算法不会漏检.

获取最危险子序列, 即得到相应的(x, y), 将其与约束进行比较, 显然用$\lfloor y\cdot(1-p)\rfloor$可以得到在$(p, k)$约束下长度为 y 的窗口允许错过截止期的次数, 记作 x', $(x'-x)/y$ 是危险系数, 根据表 1 确定任务的优先级.

表 1 任务优先级顺序

逐对比较各类任务
EDF 调度
$(x'-x)/y$ 值最小的先调度
比值相同时, y 值小的先调度
其它情况, 根据 FIFO 进行调度

在调度过程中, 对违背约束的任务作标记, 带有标记的任务实例优先执行, 当其满足截止期时清除标记.

3.2 AWCS

把计算成功率和定位最危险转折点合并为函数 FindRiskPoint, 同时扩展 CDBS 中测试序列的数据结构, TestSequence 的新数据结构如下.

```
typedef enum {
    MISS= 0;      //错过截止期
    MEET= 1;      //满足截止期
} STATUS;      //执行情况
struct TurnPoint{      //转折点
    int nTotalTask;   //本转折点后面子序列的实例总数
    int nSuccessTask;   //本转折点后面子序列的成功数
    TurnPoint * pNext;   //后继转折点
    TurnPoint * pPre;   //前驱转折点
}
struct TestSequence{//测试序列
    int nTotalTask;   //测试序列中实例总数
    STATUS arrayStatus[k];   //保存最后k次情况的数组
    TurnPoint * pTurnPoint;   //测试序列中的转折点链表
    TurnPoint * pRiskTurnPoint;   //最危险转折点指针
    double dMaxRatio;   //最大丢失率
}
```

函数 FindRiskPoint: 返回长度大于 k 的丢失率最大的子序列中长度最短的转折点的指针, 否则返回 null, 表示取序列中最后 k 个任务实例.

输入: 任务编号 i

(1) 初始化 τ_i 测试序列中的最危险转折点指针, pRiskTurnPoint ← null;

(2) 按照 τ_i 测试序列中转折点链表逆序计算丢失率, 当转折点实例总数小于 k_i 时停止, 最大丢失率记为 dR_1, 修改 pRiskTurnPoint, 使其指向具有最大丢失率且最靠近链首的转折点;

(3) 计算 τ_i 测试序列中后k_i 位组成的子序列的丢失率, 记作 dR_2;

(4) 如果 $dR_1 > dR_2$, 则 dMaxRatio ← dR_1; 否则, pRiskTurnPoint ← null, dMaxRatio ← dR_2.

AWCS 算法:

(1) 根据 3.1 节表 1 定义的优先级分配策略调度任务;

(2) 根据 τ_i 实例的执行情况, 更新任务状态: (a) 构建转折点, CreateTurnPoint(i, status); (b) 裁剪执行序列, CutDown(i); (c) 定位最危险转折点, FindRiskPoint(i); (d) 如果执行成功, 清除 τ_i 的标记; (e) 如果执行失败, 若 τ_i 的最大丢失率大于 $1-p_i$, 则对 τ_i 作标记;

(3) 转到(1).

3.3 复杂性分析

基于(p, k)约束, 假设转折点数量为 l, 就绪队列中任务个数为 n, 算法复杂度分析如下:

(1) 时间复杂度:

第一步: 计算复杂度为 $O(\log(n))$.

第二步: 创建转折点为 $O(l)$, 定位最危险转折点为 $O(l)+O(k)$, 序列裁剪为 $O(l)+O(k)$. 因此, 任务 τ_i 中每个实例调度的计算复杂度为 $O(l_i)+(O(l_i)+O(k_i))+(O(l_i)+O(k_i))=O(l_i)+O(k_i)$.

(2) 空间复杂度: 任务 τ_i 的空间复杂度为 $O(k_i)+O(l_i)$.

3.4 频繁违背约束时的近似处理

系统重度过载会导致频繁违背(p, k)约束, 使转折点个数随任务实例数增加而增加, 复杂度迅速上升, 因此需要提出简化算法, 即适当删除一些转折点. 实验分析了最危险转折点的最大编号, 表明其编号并不收敛.

由于 AWCS 在轻度过载时转折点平均个数低于 k, 因此选取 k 作为保留的转折点个数, 当转折点多于 k 时, 删除多出的转折点, 由于仅考虑 k 个窗口, 因此称为 K 窗口约束调度 (K-Window Constraint Schedule, KWCS) 算法. 实验表明 KWCS 与 AWCS 的性能相差无几, 但其在开销上有大幅度的优化.

在KWCS 中, 算法不再依赖于序列裁剪, 仅包含两个函数 CreateTurnPoint 和 FindRiskPoint.

函数 CreateTurnPoint: 构建转折点

输入: 当前任务实例是否成功, 任务编号 i

(1) τ_i 的测试序列中的任务实例总数加 1, 更新测试序列中的 arrayStatus;

(2) 如果成功, 那么 τ_i 测试序列中所有转折点实例总数加 1, 成功数加 1;

(3) 如果不成功, 那么

(a) 若 τ_i 测试序列中的第一个转折点的成功数为 0, 那么 τ_i 测试序列中所有转折点实例数加 1;

(b) 否则, 为 τ_i 测试序列新建一个转折点(实例数为 1, 成功数为 0), 放在转折点链表的链首, τ_i 测试序列中其它转折点实例数加 1;

(4) 如果转折点个数超过 k_i, 则删除链表中最后一个转折点.

函数 FindRiskPoint 的过程定义与 3.2 节相同.

KWCS 算法:

(1) 根据 3.1 节表 1 定义的优先级分配策略调度任务;

(2) 根据 τ_i 实例的执行情况, 更新任务状态:

(a) 构建转折点, CreateTurnPoint(i, status);

(b) 定位最危险转折点, FindRiskPoint(i);

(c) 如果执行成功, 清除 τ_i 的标记;

(d) 如果执行失败, 若 τ_i 的最大丢失率大于 $1-p_i$, 则对 τ_i 作以标记;

(3) 转到(1).

KWCS 算法的复杂度分析如下:

(1) 时间复杂度:

第一步: 计算复杂度是 $O(\log(n))$;

第二步: 任务 τ_i 每个实例的复杂度为 $O(k_i)+O(k_i)=O(k_i)$;

(2) 空间复杂度: 任务 τ_i 的空间复杂度为 $O(k_i)$.

当任务模型确定后, KWCS 算法的复杂度是 $O(1)$的.

4　算法特性

4.1　公平性和区分性

AWCS 的目标是使每类任务的区间内成功率尽量接近目标成功率 p, 根据区间最小成功率与 p 的距离调度, 进而保证所有任务的区间最小成功率与 p 的距离基本相同而达到公平.

显然 AWCS 提供的公平性是建立在不同 QoS 上的, 使用(p, k)描述 QoS 需求, 利用成功率和最小区间长度区分 QoS, AWCS 算法根据区间最小成功率与 p 的距离进行调度, 当距离相同时, 会进一步根据区间长度进行区分, 因此调度时会偏向于具有较高 p 值和较小 k 值的任务, 从而保证不同任务之间 QoS 的差别.

AWCS 提供了一种既公平又有差别的服务, 这种服务能够在系统面临重度过载时缓慢地退化.

4.2　时延分析

任务实例的时延是指从任务实例产生到完成的时间, 不必考虑丢失的任务实例. 任务的时延需要考虑丢失实例带来的影响, 本文中采用如下定义.

定义 **3**　任务 τ_i 的时延 R_i 由式(1)定义:

$$R_i=f_{s_{j+1,i}}-r_{s_{j,i}+1},\ j\geqslant 0,\ s_{0,i}=1 \text{ and } 1\leqslant i\leqslant N \quad (1)$$

其中 $s_{j,i}$表示任务 τ_i 的第 j 个满足截止期的任务实例, f 表示完成时间, r 表示启动时间, N 表示任务数量.

任务的时延与任务的区间最小成功率的关系十分紧密, 为了给出一般化的任务时延分析, 定义成功率偏离度 ε, 并根据 p 和 ε给出任务时延的上界.

定义 **4**　假设区间最小成功率 p', 目标成功率是 p, 那么成功率偏离度 $\varepsilon=p-p'$.

定理 **3**　在 AWCS 调度下, 任务 τ_i 的最大时延不超过$(\lfloor k_i(1-p_i+\varepsilon_i)\rfloor+1)T_i$.

证明　根据成功率偏离度的定义, 任务 τ_i 在任意长度不小于 k 的窗口内的成功率不小于$p_i-\varepsilon_i$, 那么窗口内丢失截止期的任务个数不超过$\lfloor k_i(1-p_i+\varepsilon_i)\rfloor$, 即不会出现$\lfloor k_i(1-p_i+\varepsilon_i)\rfloor+1$ 个任务实例连续丢失截止期的情况, 因此任务 τ_i 的最大时延不超过$(\lfloor k_i(1-p_i+\varepsilon_i)\rfloor+1)T_i$. 证毕.

确定最小区间长度后, 通过模拟可以得到各种调度算法的区间最小成功率, 进而得出其成功率偏离度 ε, 假设任务的 QoS 需求考虑的最小区间长度为 k, 可以得到推论 1.

推论 **1**　应用 QoS 保证的调度算法, 任务 τ_i 的最大时延不超过$(\lfloor k_i(1-p_i+\varepsilon_i)\rfloor+1)T_i$.

5　实验

5.1　复杂度实验

AWCS 的复杂度与转折点个数密切相关, 由于 AWCS 在重度过载时复杂度迅速提高. 因此需要简化, 通过适当删除转折点来降低复杂度, 为此模拟最危险转折点的最大编号的变化. 考虑同构的实验环境: 随机选定 100 组任务模型, 每组包含 20 个任务, 固定任务计算时间为 1, 随机分配任务的周期 T, 使其在$[1, 100]$ 内均匀变化, 同时使每组任务的利用率 u 在$[1.1, 2.0]$ 内变化, 固定 $k=10$, 目标成功执行率 $p=70\%$, 仿真时间为 20000. 图 1 给出最危险转折点最大编号变化图, 显然随着利用率的增加, 最危险转折点的最大编号不收敛.

5.2　性能比较

本文采用以下性能评估参数:

(1) (m, k) 动态失效率: 统一使用 (m, k) 约束进行动态失效率的统计;

(2) 最小成功率: 每类任务总体成功率的最小值;

(3) 区间最小成功率: 记录所有长度不小于 k 的区间最小成功率, 用于模拟成功率偏离度;

(4) 时延: 根据式(1)计算任务的时延.

5.2.1 同构实验

首先考虑同构条件, 即所有任务的约束都相同. 随机选定 100 组任务模型, 每组模型包含 20 个任务, 固定任务的计算时间为 1, 随机分配任务的周期 T, 使其在 $[1, 100]$ 内均匀变化, 同时使得每组任务的利用率 u 在 $[1.0, 2.0]$ 内变化, 固定 $k=10$, 目标成功执行率 $p=70\%$. DBP 中任务约束为 $(7, 10)$; DWCS 中任务约束为 $(3, 10)$; HAS 参数设置如下: 全局成功率 $Q=0.7$, 最大连续失败次数 $F=3$, 启发式参数 $a=b=0.5$. 仿真时间为 20000.

(1) (m, k) 动态失效率和最小成功率: 由图 2 可以看出, 在动态失效率方面, 轻度过载情况下 DBP 的动态失效率最小, 当重度过载时, 各类算法的动态失效率相当; 在最小成功率方面, DBP 在重度过载情况下出现饿死现象, HAS 主要是考虑全局成功率, 因此其表现最好, 而 AWCS(KWCS) 的性能优于 CDBS 和 DWCS. AWCS 和 KWCS 算法的性能几乎相同.

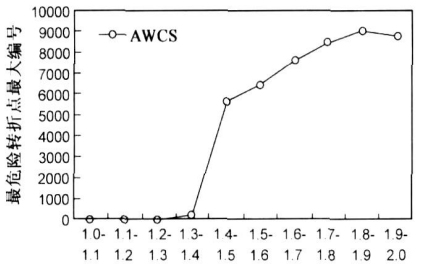


图 1 最危险转折点最大编号变化图

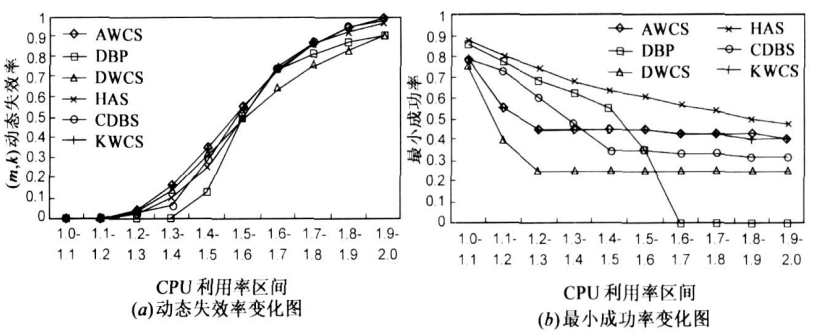


图 2 同构环境下性能变化图

(2) 区间最小成功率: 图 3 给出区间最小成功率变化图, 可以发现尽管 HAS 在全局最小成功率方面表现最好, 但是在某一长度大于最小区间长度的窗口内成功率出现0的情况, 说明其截止期满足与丢失的分布不均匀, AWCS 和 KWCS 的性能几乎相同, 优于其他算法, 说明在 AWCS(KWCS) 调度下, 区间最小成功率最大, 因此成功率偏离度最小, 从而任务最大延时上界最小, 也说明其任务实例满足截止期和丢失截止期的分布比较均匀.

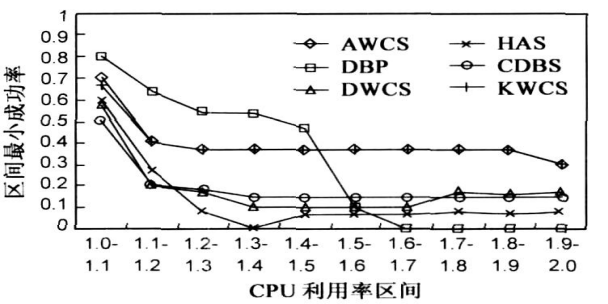


图 3 区间最小成功率变化图

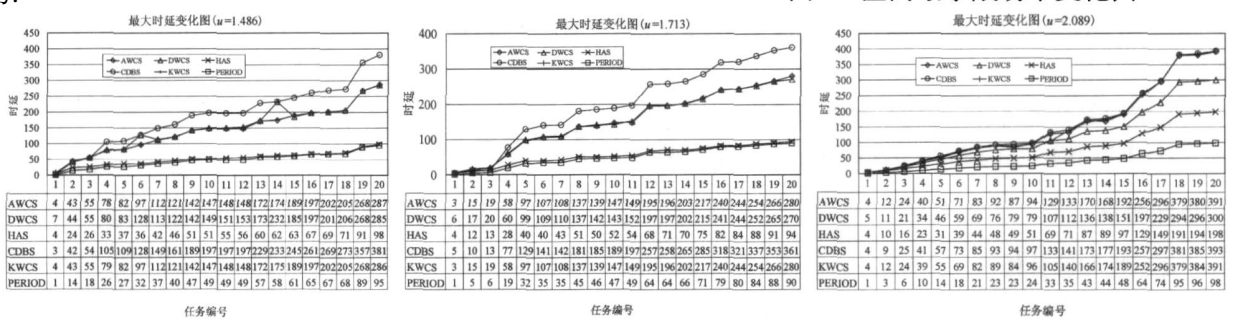


图 4 CPU 利用率在 [1.4,1.5]、[1.7,1.8]、[2.0,2.1] 区间内的最大时延变化图

(3) 时延: 由于 DBP 算法已经出现饿死的情况, 因此在时延比较中不再与 DBP 进行比较. 在 $[1.4, 1.5]$、$[1.7, 1.8]$、$[2.0, 2.1]$ 三个 CPU 利用率区间内各随机抽取一组任务. 如图 4(PERIOD 表示任务周期), 给出三个区间的抽样结果, 可以发现 AWCS(KWCS) 对于每类任务的最大时延与周期之比基本是常数, 说明其对具有相同 QoS 需求的任务的公平性; 而 HAS 对大周期任务具有偏向性.

5.2.2 异构实验

在异构实验中, 考虑五类约束: $(9, 10)$、$(3, 4)$、$(1, 2)$、$(1, 3)$ 和 $(1, 4)$[6], 每类算法的参数配置如表 2 所示.

首先需要区分 CPU 利用率 u 和等效 CPU 利用率 $u(m, k)$, 两者分别定义如下:

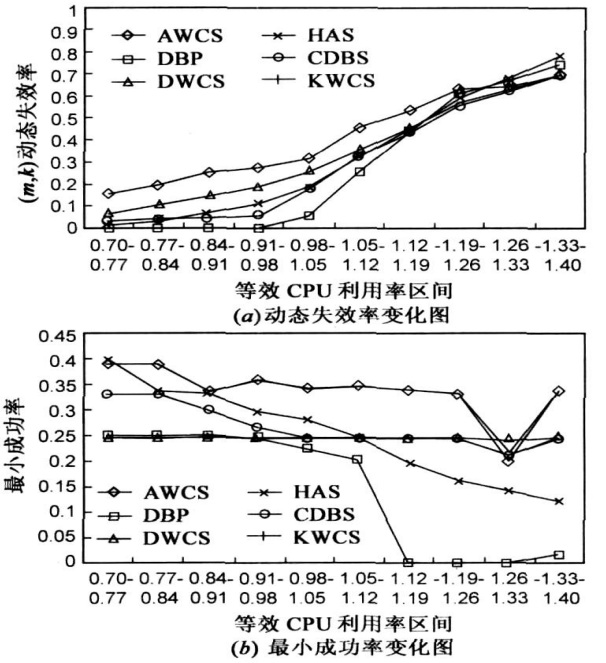
$$u=\sum_{i=1}^{N}\frac{C_i}{T_i} \text{ 和 } u(m, k)=\sum_{i=1}^{N}\frac{m_i C_i}{k_i T_i}$$

表 2　调度算法参数配置表

	DBP		DWCS		CDBS		HAS		AWCS(KWCS)	
	m	k	x	y	m	p	F	Q	p	k
(9, 10)	9	10	1	10	1	0.9	1	0.9	0.9	10
(3, 4)	3	4	1	4	1	0.75	1	0.75	0.75	4
(1, 2)	1	2	1	2	1	0.5	1	0.5	0.5	2
(1, 3)	1	3	2	3	2	0.33	2	0.33	0.33	3
(1, 4)	1	4	3	4	3	0.25	3	0.25	0.25	4

在同构实验环境下, CPU 利用率和等效 CPU 利用率的变化相同, 因此可以用 CPU 利用率来代表等效 CPU 利用率, 而在异构实验环境下, 二者变化不同, 因此必须根据等效 CPU 利用率来设计实验. 随机选定 100 组任务模型, 每组包含 20 个任务, 固定计算时间为 1, 约束从五类约束中随机选取, 随机分配任务模型的周期 T, 使其在$[1, 100]$内均匀变化, 同时使得每组模型的等效 CPU 利用率在$[0.7, 1.4]$内(与同构实验中的等效 CPU 利用率对应)变化, 根据所选约束设定调度算法的参数. 仿真时间为 20000.

(1)(m, k)动态失效率和最小成功率: 由图 5 可以看出, 在动态失效率方面, AWCS(KWCS)在轻度过载情况下动态失效率偏高, 而在重度过载情况下与其它算法相当. 其原因在于 k 值小的任务对系统的干扰, 例如对于两个任务, 一个约束为(1, 2), 另一个为(9, 10), 两个任务的执行序列分别是 011 和 0111111111, 根据算法两任务$(x' - x)/y$ 相等, 则由于第一个序列比较短, 因此优先调度第一个任务, 而实际情况是第一个任务失败后序列为 0110, 不会引起动态失效, 而第二个任务失败后序列为 01111111110 却引起动态失效, 从而导致动态失效率增加, 而在重度过载情况下, 这种影响减少. 在最小成功率方面, 结果与同构情况下相似.

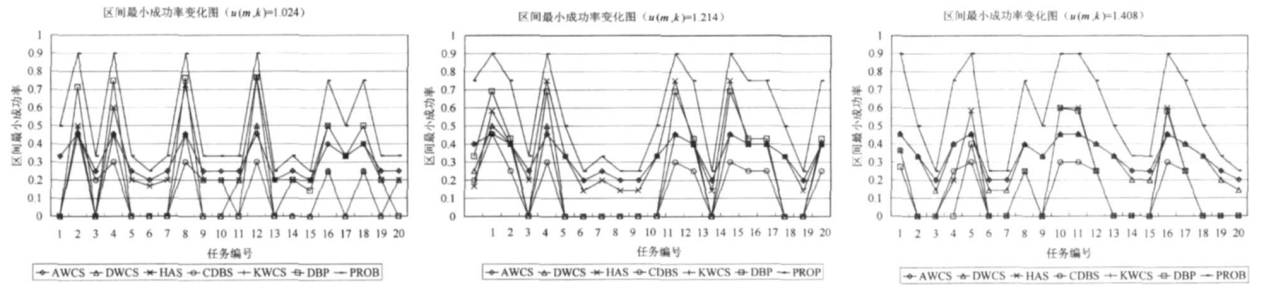


(a) 动态失效率变化图

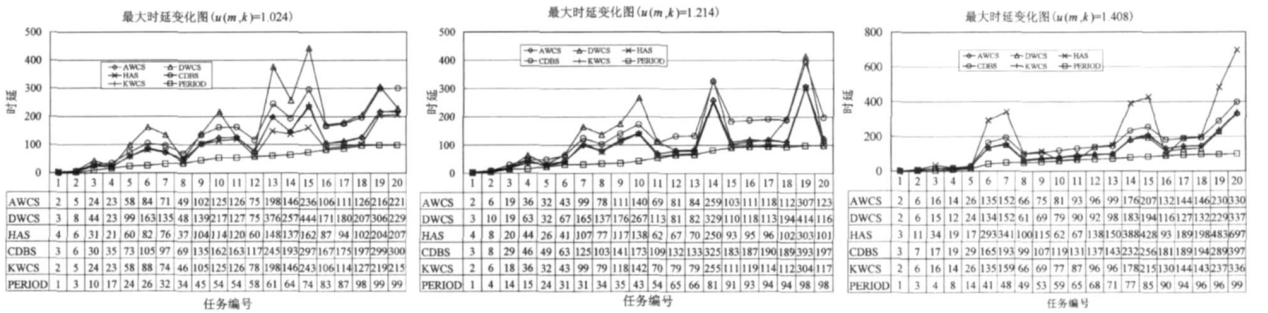
(b) 最小成功率变化图

图 5　异构环境下性能变化图

(2)区间最小成功率和时延: 在$[0.98, 1.05]$、$[1.19, 1.26]$、$[1.4, 1.47]$三个等效 CPU 利用率区间内各随机抽取一组任务, 图 6 给出三组任务的区间最小成功率(PROB 表示目标成功率)和最大时延变化图(PERIOD 表示任务周期), 与同构实验类似, 在时延比较时不考虑 DBP 算法, 从图中可以看出 DWCS、CDBS、DBP 和 HAS



(a) 等效 CPU 利用率在 [0.98,1.05]、[1.19,1.26]、[1.40,1.47]区间内的区间最小成功率变化图



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AWCS	2	5	24	23	58	84	71	49	102	125	126	75	198	146	236	106	111	126	216	221
DWCS	3	8	44	23	99	163	135	48	139	217	127	75	376	257	444	171	180	207	306	229
HAS	4	6	31	21	60	82	76	37	104	114	120	60	148	137	162	87	94	102	204	207
CDBS	3	6	30	35	73	105	97	69	135	162	163	117	245	193	297	167	175	197	299	300
KWCS	2	5	24	23	58	88	74	46	105	125	126	78	198	146	243	106	114	127	219	215
PERIOD	1	3	10	17	24	26	32	34	45	54	54	58	61	64	74	83	87	98	99	99

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AWCS	2	6	19	36	32	43	99	78	111	140	69	81	84	259	103	111	118	112	307	123
DWCS	3	10	19	63	32	67	165	137	176	267	113	81	82	329	110	118	113	194	414	116
HAS	4	8	20	44	26	41	107	77	117	138	62	67	70	250	93	95	96	102	303	101
CDBS	3	8	29	46	49	63	125	103	141	173	109	132	133	325	183	187	190	189	393	197
KWCS	2	6	18	36	32	43	99	79	118	142	70	79	79	255	111	119	114	112	304	117
PERIOD	1	4	14	15	24	31	31	34	35	43	54	65	66	81	91	93	94	94	98	98

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AWCS	2	6	16	14	26	135	152	66	75	81	93	96	99	176	207	132	144	146	230	330
DWCS	2	6	15	12	24	134	152	61	69	79	90	92	98	183	194	116	127	132	229	337
HAS	3	11	34	19	17	293	341	100	115	62	67	138	150	388	428	93	189	198	483	697
CDBS	3	7	17	19	29	165	193	99	107	119	131	137	143	232	256	181	189	194	289	397
KWCS	2	6	16	14	26	135	159	66	69	77	87	96	96	178	215	130	144	143	237	336
PERIOD	1	3	4	8	14	41	48	49	53	59	65	68	71	77	85	90	94	96	96	99

(b) 等效 CPU 利用率在 [0.98,1.05]、[1.19,1.26]、[1.40,1.47]区间内的最大时延变化图

图 6　区间最小成功率和最大时延变化图

都出现了区间内饿死的现象,而 AWCS(KWCS)根据 QoS 需求的不同,提供了适当的退化.

6 总结

本文从可变区间的丢失率保证问题出发,研究在过载情况下的公平与差别服务.本文提出(p,k)约束并用其描述任务的 QoS 需求,设计基于(p,k)约束的 AWCS 算法,分析了 AWCS 算法的复杂度,并根据其在重度过载情况下复杂度剧增而不适合调度的情况,对其进行简化,提出 KWCS,实验表明 KWCS 与 AWCS 性能相当.通过分析 AWCS(KWCS)提供的公平性和差别性,进一步定义成功率偏离度,并给出应用 QoS 保证的调度算法的时延上界的通用表示方法.最后,将算法与其它弱硬实时算法比较,结果表明 AWCS(KWCS)在重度过载情况下优于其它算法,能够使任务的 QoS 缓慢地退化,提供一种既公平又有差别的服务.

参考文献:

[1] Zheng Wang. Internet QoS: Architecture and Mechanisms for Quality of Service[M]. San Francisco: Morgan Kaufmann Publishers, 2001. 1-3.

[2] El-gendy M A, Bose A, Shin K G. Evolution of the Internet QoS and support for soft real-time applications[J]. Proceedings of the IEEE. 2003, 91(7): 1086-1104.

[3] Chou C, Shin K G. Statistical real-time channels on multiaccess bus networks[J]. IEEE Transactions on Parallel and Distributed Systems. 1997, 8(8): 769-780.

[4] Cho J, Shin H. Heuristic scheduling for multimedia streams with firm deadlines[A]. Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications[C]. Washington, DC, USA: IEEE Computer Society, 1997. 67-72.

[5] Koren G, Shasha D. An approach to handling overloaded systems that allow skips[A]. Proceedings of the 16th IEEE Real-Time Systems Symposium[C]. Pisa: IEEE Computer Society, 1995. 110-119.

[6] Hamdaoui M, Ramanathan P. A dynamic priority assignment technique for streams with (m, k)-firm deadlines[J]. IEEE Transactions on Computers, 1995, 44(4): 1443-1451.

[7] Wedde H, Lind J. Building large, complex, distributed safety-critical operating systems[J]. Real-Time Systems, 1997, 13(3): 277-302.

[8] Bernat G, Cayssials R. Guaranteed on-line weakly-hard real-time systems[A]. Proceedings of the 22nd IEEE Real-Time Systems Symposium[C]. London: IEEE Computer Society, 2001. 25-35.

[9] Gang T, Min Y F, Sheng L Y. Scheduling algorithms based on weakly hard real-time constraints[J]. Journal of Computer Science and Technology, 2003. 18(6): 815-821.

[10] Richard W, Yuting Z, Karsten S, Christian P. Dynamic window-constrained scheduling of real-time streams in media servers[J]. IEEE Transactions on Computers, 2004, 53(6): 744-759.

[11] Yuting Z, Richard W, Xin Q. A virtual deadline scheduler for window-constrained service guarantees[A]. Proceedings of the 25th IEEE Real-Time Systems Symposium[C]. Washington, DC, USA: IEEE Computer Society, 2004. 151-160.

[12] 陈积明, 宋叶琼, 孙优贤. 弱硬实时系统约束规范研究[J]. 软件学报, 2006, 17(12): 2601-2608.
Jiming C, Zhi W, Yeqiong S, Youxian S. Research on constraint specification of weakly hard real-time system[J]. Journal of Software, 2006, 17(12): 2601-2608. (in Chinese)

[13] 吴彤, 金士尧, 刘华锋, 陈积明. 基于裁剪的弱硬实时调度算法[J]. 软件学报, 2008, 19(7).
Tong W, Shiyao J, Huafeng L, Jiming C. A weakly hard real-time schedule algorithm based on cutting down[J]. Journal of Software, 2008, 19(7). (in Chinese)

[14] Guillem B, Alan B, Albert L. Weakly hard real-time systems[J]. IEEE Transactions on Computers, 2001, 50(4): 308-321.

作者简介:

吴 彤 男,1979 年 9 月出生于辽宁省沈阳市,博士生,主要研究领域为弱硬实时调度.
E-mail: treewutong@163.com

金士尧 男,生于 1937 年,博士生导师,主要研究领域为实时系统、分布计算与仿真.
E-mail: syjin1937@163.com

陈积明 男,生于 1978 年.博士,副教授.主要研究领域为弱硬实时调度、无线传感器/执行器网络.(本文通信作者)
E-mail: jmchen@ieee.org