一种同构多处理机动态实时调度算法①

陆小双, 帅建梅

(中国科学技术大学 信息科学技术学院, 合肥 230027)

摘 要: 本文提出一种新型线性复杂度多处理机实时任务启发式调度算法, 利用并行技术为动态实时系统提供较优解. 使用大量存在可行调度的任务集合测试多处理机实时任务调度算法的性能, 分析了几种主要参数对调度成功率的影响. 实验表明新调度算法调度成功率较高, 适用于不完全知晓任务参数的动态多处理机实时系统.

关键词: 实时调度; 多处理机; 启发式算法; 并行; 调度成功率

Novel Dynamic Real-Time Scheduling Algorithm for Homogeneous Multiprocessor Systems

LU Xiao-Shuang, SHUAI Jian-Mei

(School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: This paper introduces a novel heuristic algorithm with linear computational complexity for multiprocessor systems. It provides suboptimal solutions for dynamic real-time systems by parallelization of tasks. The performance of scheduling algorithms for multiprocessor platforms is compared by a large quantity of scheduable task sets. Experiments show that the new scheduling scheduling algorithm has a higher success ratio and is appropriate for dynamic real-time systems without a complete prior knowledge of task parameters.

Key words: real-time scheduling; multiprocessor; heuristic algorithms; parallelization; success ratio

1 绪论

实时系统在通信系统、车辆电子、医疗图像、航天航空等领域都有广泛应用, 其正确性不仅取决于计算逻辑, 还取决于完成时间[1]. 能被系统调度和执行的工作单元称为作业; 共同提供某种功能的一组作业称为任务; 若没有混淆, 下文将不予区分. 常见的实时任务调度算法包括时钟驱动, 加权轮换, 优先级驱动等. 优先级驱动调度可以按照静态或动态、可抢占或不可抢占等进行分类. 静态优先级调度为每个任务中的所有作业分配相同的优先级; 而动态优先级调度为每个任务中的不同作业分配不同的优先级. 如果作业的执行在任何时候都能被其他作业中断, 随后又可以恢复执行, 称作业是可抢占的; 如果一个作业必须从头至尾地运行, 不能被中断, 称作业是不可抢占的.

2 相关研究

目前, 单处理机调度已有大量研究. 常见的静态优先级调度算法有速率单调(RM)和(相对)时限单调(DM). 最早(绝对)时限优先(EDF)和最小空闲时间优先(LLF)则是动态优先级算法.

RM 算法简单易于实现并且运行开销小, 系统超载时也能保持较好的鲁棒性. 但 RM 算法要求事先知道任务的信息, 包括周期和时限, 其处理机的利用率也较为低下. 另外, 该算法还需要周期和相对时限相等等假设. 如果任务的相对时限和周期成正比, 那么 DM 算法与 RM 算法等价. 相对时限任意时, DM 算法表现要好, 因为 RM 不能找到可行调度的任务集合, DM 可能找到可行调度. EDF 算法和 LLF 算法不需要完全知晓作业所有的参数, 并且能获得较高的处理机利用率, 但它们不能应付系统的超载, 一个作业错过

① 基金项目:国家科技支撑计划课题 (2011BAH11B01);中科院先导专项“感知中国” (XDA06030900)
收稿时间:2013-05-09;收到修改稿时间:2013-05-27

时限可能导致一系列的作业错过时限. 另外, 两者对于作业不可抢占或是多处理机系统都是非最优的.

如果事先不能完全知晓任务的时限、计算时间以及开始时间, 对于有两个或者两个以上处理机的系统, 不存在最优的实时调度算法[2]. 该消极结论催生了使用启发式算法解决实时多处理机系统任务调度问题的需求. 对此, 人们则提出了近视算法[3]、并行近视算法[4]和节约算法[5]等.

文献[3]表明, 动态调度实际是搜索问题. 搜索空间为一棵树: 中间结点对应局部调度, 叶子结点对应完整调度. 启发式调度算法从根节点开始(空调度)利用下一层节点扩展当前调度, 搜索过程使用启发式函数 . 在扩展局部调度时, 算法会判断当前局部调度是否“强可行”. 若“强可行”, 则用 函数值最小的作业进行扩展. 若非“强可行”的, 则回退到上一层, 并选择函数值次小的作业进行扩展. 文献[3]提出的近视算法与普通搜索算法的不同在于扩展过程, 前者会考虑所有剩余作业, 而后者只考虑前若干个作业.

文献[4]在近视算法的基础上提出了利用作业并行性的新算法: 当不能找到满足作业时限的处理机时, 就并行执行作业.

文献[5]显明, 近视算法选择某一个作业扩展时, 会从可满足作业时限的所有处理机中挑选最早可用的, 这种方式可能延迟未被调度任务的开始时间. 因此提出节约算法: 选择可用时间最晚并且能够满足作业时限的处理机.

文献[6]认为, 扩展当前调度每次只考虑一个作业, 调度预测性对于整个系统的作业来说是有限的, 调度成功率有待提高. 从而提出动态分批优化算法: 每次选取一批作业来扩展当前调度. 该算法存在的问题是: 批量扩展时, 不能满足时间约束的作业将会在下次扩展中重新被考虑, 这增大其不能被成功调度的可能性. 更糟糕的是, 部分作业不能按时完成很可能直接导致算法的回退.

3　分批并行多处理机动态实时调度

3.1 作业分批问题描述

本文主要介绍一种利用任务并行性的非抢占多处理机分批并行动态调度算法. 该算法适用于非周期性任务、任务可并行处理、任务间没有直接约束的实时系统. 算法采用类似于滑动窗口的结构批量扩充当前调度. 对于滑动窗口内不能找到处理机满足其时限要求的作业, 则试图利用作业的并行性, 从而减少回退, 增大找到可行调度的可能性. 本文使用集中调度模型: 所有的作业都到达称为调度器的处理机(不属于下文讨论的处理机集合). 然后由该调度器将作业分发至其他的处理机执行.

对于同构多处理机系统, 处理机集合记作 $\prod=\{\pi_1,\pi_2,...,\pi_l\}$. 其他各类的资源记作 $R=\{R_1,R_2,...,R_m\}$, 每种资源有一个实例. 使用 $J=\{\tau_1,\tau_2,...,\tau_n\}$ 表示作业集合. 作业 τ_i 由元组 (a_i,r_i,e_i^j,d_i,V_i) 描述, 其中 $1\le i\le n$ 且 $1\le j\le l$. a_i 是到达时间; r_i 是就绪时间; e_i^j 是作业 τ_i 在 j 个处理机上的最坏并行计算时间; d_i 是绝对时限; 矩阵 $V_i=[u_1\ \ u_2\ \ ...\ \ u_m]$ 是作业的资源需求 ($u_k=0/1/2$). u_k 为资源 R_k 的使用模式: 0 表示不需要该资源; 1 表示以共享模式; 2 表示独占模式. 作业不会同时以共享模式和独占模式请求同一资源. 使用 $REAT^s$ 矩阵和 $REAT^e$ 矩阵分别代表在共享模式和独占模式下的资源最早可用时间.

$$REAT^s=[REAT_1^s\ \ REAT_2^s\ \ ...\ \ REAT_m^s]$$

$$REAT^e=[REAT_1^e\ \ REAT_2^e\ \ ...\ \ REAT_m^e]$$

每次扩展当前调度时, 算法都通过 $REAT^s$ 和 $REAT^e$ 重新计算剩余作业的最早开始时间. 作业 τ_j 在处理机 π_i 上的最早开始时间 $EST(\pi_i,\tau_j)$ 可记为

$$\max\{\tau_j.r,\pi_i,\max_{y\in\{1,2,...,m\},u\in\{s,e\}}\{REAT_y^u\}\}$$

定义作业 τ_j 分配在处理机 π_i 上运行的成本函数为 $C(\pi_i,\tau_j)=\tau_j.d+w\cdot EST(\pi_i,\tau_j)$. 式中 w 是权值. 下面构造成本矩阵 C 和分配矩阵 X. 每个处理机只能做一个作业或者空闲, 每个作业只能由一个处理机完成. $l\le n$ 时, 所有处理机都不空闲, 所以, $\sum X_{i,j}=1$ 其中 i 给定, j 变化; 因为存在不能被处理的作业, 所以 $\sum X_{i,j}\le 1$, 其中 j 给定, i 变化. 问题变成, 找到一个分配矩阵, 使得总成本 $\sum\sum(C(\pi_i,\tau_j)\cdot X_{i,j})$ 最小, 即 J 中的 l 个作业 $\tau_{y_1},\tau_{y_2},...,\tau_{y_l}$ ($\{y_1,y_2,...,y_l\}\subseteq\{1,2,...,n\}$)分别被分配在处理器 $\pi_1,\pi_2,...,\pi_l$ 运行是最优的. 同理, $n\le l$ 时, 存在空闲的处理机, 所以 $\sum X_{i,j}\le 1$, 其中 i 给定, j 变化; 因为所有作业都能被处理, 所以 $\sum X_{i,j}=1$, 其中 j 给定, i 变化. 问题变成, 找到一个分

配矩阵, 使得总成本$\sum\sum(C(\pi_i,\tau_j)\cdot X_{i,j})$最小, 即$J$中的 n 个作业$\tau_1,\tau_2,...,\tau_n$分别被分配在处理器$\pi_{y_1},\pi_{y_2},...,\pi_{y_n}$($\{y_1,y_2,...,y_n\}\subseteq\{1,2,...,l\}$)运行是最优的.

上述问题的本质是分配问题: 给定代理集合 A、任务集合 T 以及定义在 $A\times T$ 上的成本函数 C, 求一种分配方式使得总成本$\sum C$最小. 当代理数 m 等于任务数 n 时, 可直接使用匈牙利算法求解. 如果$m\neq n$, 可添加虚设代理或者虚设任务. 文献[7]提出了免于使用虚设代理($m<n$)的 Kumar 算法. 注意到, 如果一次将多个任务同时分配给某一处理机, 后续的可行性检查会变得较为复杂, 因此可将多余任务留待以后分配. Kumar 算法修改如下.

1.构造成本矩阵.

2.如果$m=n$, 直接使用匈牙利算法求解, 算法结束. 否则转步骤 3.

3.计算每列的和, 结果存于序列 *SumColumn*.

4.从 *SumColumn* 中选取前 m 个和最小的列(任务)构造新问题.

5.使用匈牙利算法解决新问题, 新问题的解就是原问题的最优解. 算法结束.

如果通过 Kumar 变体算法求得的解中, 存在不能满足作业时限的分配, 则试图对作业进行并行处理.

3.2 分批并行动态调度算法

下面阐述一种新的适用于同构多处理机系统、任务为非周期性且相互独立的动态多处理机实时任务调度算法. 设分批窗口的大小为 N; 作业的最大并行度为 *SplitMax*; 允许的最大回退次数是 *BackTrackMax*; 处理机集合为$\prod=\{\pi_1,\pi_2,...,\pi_l\}$; 资源集合为$R=\{R_1,R_2,...,R_m\}$; 作业集合为$J=\{\tau_1,\tau_2,...,\tau_n\}$. 令 T 表示算法每次实际考虑的作业数目$T=\min\{N,|J|\}$. 算法流程如下.

① 按最早时限优先为作业集合 J 中所有作业分配优先级.

② 将作业集合 J 中的作业按照各自优先级以非递增的方式排列. 同时, 设置集合 S 与集合 S*为空, $BackTrack=0$.

③ 从作业集合 J 中依次选取作业到集合 S 中, 直到作业集合 J 中已经不存在作业或者集合 S 中的作业数目达到 N. 记集合 S 为$\{\tau_{x_1},\tau_{x_2},...,\tau_{x_T}\}$($T\le N$, $\{x_1,x_2,...,x_T\}\in\{1,2,...,n\}$). 此外, 选取的作业还需要满足如下条件. 若存在 i 和 j($i\in\{1,2,...,T\}$, $j\in\{1,2,...,m\}$), 使得$\tau_{x_i}\cdot u_j\neq 0$, 则对任意的 k($k\in\{1,2,...,T\}$, $k\neq i$), 满足$\tau_{x_k}\cdot u_j=0$. 即当集合 S 中的某一作业对资源 R_j 有访问需求时, 集合 S 中的其它作业不能有访问该资源的需求. 更新$J=J-S$.

④ 构造集合 S 中各个作业在所有处理机上的成本函数 $C(\pi_i,\tau_{x_j})=\tau_{x_j}.d+w\cdot EST(\pi_i,\tau_{x_j})$, 其中 $i\in\{1,2,...,l\}$ 且 $j\in\{1,2,...,T\}$.

⑤ 利用修改的 Kumar 算法为作业集合 S 找到某种最优分配.

⑥ 检查作业集合 S 的最优分配的强可行行. 如果分配中存在作业τ使得$EST(\pi_i,\tau)+\tau.e>\tau.d$成立, 即作业的时限不能满足, 则说明为作业集合 S 求解的调度是不可行的.

⑦ 如果作业集合 S 的最优分配不可行, 则使用并行近视算法搜索一种局部调度. 如果这样的局部调度不存在, 将集合 S*设置为作业集合 S 中所有哪怕是并行都不能被调度的作业. 令$J=J\cup(S-S^*)$, $S=S^*$, $S^*=\{\}$, $BackTrack=BackTrack+1$, 回退至上一层调度, 转步骤 4, 直到回退次数已经超过最大值 *BackTrackMax* 或者已经没有再回退的可能. 此时调度失败, 算法结束.

⑧ 如果作业集合 S 的最优分配可行, 则更新相关数据结构, 将已经用于扩展局部调度的作业了从 S 中剔除, 重复步骤 3 至 8. 直到找到了一个完全可行的调度. 此时将调度分发至各个处理机. 算法结束.

假设当前任务队列中有 n 个作业, 调度选取作业的窗口大小为 N. 系统中拥有 l 个处理机和 m 个资源. 由于算法的每一次都要构造成本矩阵, 其复杂度为$O(l\cdot|S|)$. 而要对批量窗口使用 Kumar 变体算法, 则相应的复杂度为$O(\max\{l,|S|\}^3)$. 当求得的解不能满足某些作业的时限时, 需要考虑窗口中作业的并行性并采用近视算法找到某种局部调度使得相关都可行, 对应的复杂度为$O(|S|)$. 因此, 算法的总时间复杂度为$O((l\cdot|S|+\max\{l,|S|\}^3+|S|)\cdot n/|s|)$. 易见, 算法复杂度跟窗口大小紧密相关. 如果窗口大小偏大将使得求解问题变得缓慢. 通常窗口大小以及处理机数目为常数, 且远小于作业数目 n. 因此, 总复杂度可简记为$O(c\cdot n)$, 其中 c 为一常数, 即算法复杂度是线性的, 与近视算法、节约算法和批优化调度算法相同.

4 仿真和实验

4.1 评价指标

为了解调度的算法的效率, 需要对其进行仿真和实验. 文献[3]指出满足任务的时限是实时任务调度最为重要的目标之一. 因此, 可使用算法找到的可行调度任务集合数目与总任务集合数目的比值作为指标.

4.2 实验方法

为研究调度成功率, 需要生成能够找到可行调度的任务集合. 因此, 需要开发任务集合生成器. 这个任务集合生成器需要满足下列要求: 1.能够生成大量的任务并且存在可行调度; 2.能保证最大的处理机利用率. 任务集合生成器需要考虑的主要参数如表 1 所示.

表 1　仿真的参数与含义

参数	含义
l	处理机数目
MaxC	最大计算时间
MinC	最小计算时间
R	松弛参数，描述作业时限的紧急程度
UsedP	需要访问资源的概率
SharedP	以共享方式访问资源的概率
BackTrackMax	允许最大的回退次数
w	成本函数的权值参数
N	窗口大小
ScheduleLength	调度长度

作业在一个处理机上的计算时间在区间 $[MinC, MaxC]$ 之中以等概率的方式产生. 作业 τ_i 的时限在区间 $[ShortestC, (1+R)\cdot ShortestC]$ 之中也以等概率的方式产生, 其中 *ShortestC* 是所有任务的最短计算时间. 任务对资源的需求通过 *UsedP* 和 *SharedP* 生成. 作业 τ_i 在 j 个处理机上的计算时间是 e_i^j, 其中 $j \geq 2$, $e_i^j = \lfloor e_i^{j-1} \cdot (j-1)/j \rfloor + 1$. 这是由于任务被分割得越多, 各部分之间的通信和同步的代价就越大, 即对于任意的 j 和 k, 有 $j \cdot e_i^j < k \cdot e_i^k$, 其中 $j < k$. 任务集合生成器将根据上述各类参数生成包含大量任务的集合, 直到达到调度长度 *ScheduleLength*.

实验中, 主要考虑四种因素对调度成功率的影响: 时限的紧急程度、使用资源的概率、允许的最大回退次数、以及分批窗口(对应于近视算法的可行性检查窗口)的大小. 实验为每种情形进行 5 组仿真, 每组仿真使用任务集合生成器构造的 400 个任务集合, 然后使用这些任务集合对三种调度算法进行研究.

另外, 若将成本函数中的 *w* 设定的很小, 极端情况下 *w*=0, 此时近视算法、并行算法以及节约算法将局部退化成 EDF. 另一方面, 如果 *w* 的值过高将会造成作业的时限得不到充分考虑, 直接导致的后果便是大量作业错过时限. 根据以往的研究经验, *w*=4 是一个妥当的方案. 仿真中处理机数目 *l* 为 8, 调度长度固定在 800, 每个任务集合大约包含 190 个任务.

4.3 结果对比和分析

如前所述, 松弛参数 *R* 表示作业时限的紧急程度, 如果值越小说明时限较早的作业所占的比例就越大. 因此, 系统在 *R* 较小时调度成功率相对来说较低. 图 1 表示不同时限紧急程度对任务集合进行仿真实验的结果. 实验中 *N*、*w*、*UsedP*、*SharedP* 和 *BackTrackMax* 分别为 7、4、0.2、0.5、9. 从图可以看出新算法和已有算法的调度成功率都随松弛的增大而增大. 当松弛参数处于较低的水平时, 大量作业的时限比较紧急, 造成调度成功率的低下. 此时新算法占有大的优势. 而松弛参数变得足够大时, 老的算法的调度成功率也有提升, 这是随着时限靠后作业所占比例的增长, 它们也更加容易找到可行调度了.

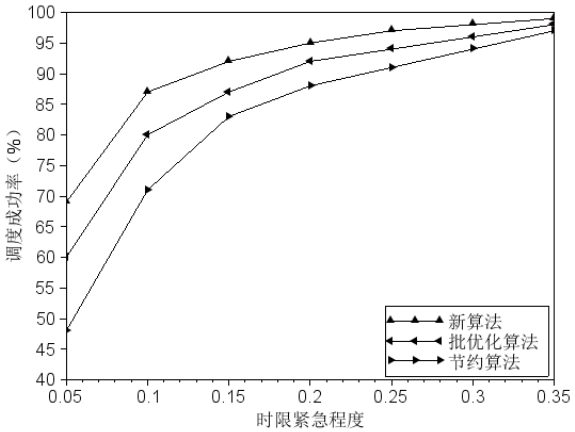

图 1　时限的紧急程度对调度成功率的影响

显然, 当作业不依赖除处理机之外的资源时, 找到可行调度将更加容易, 这是因为系统避免了可能的资源竞争的原故. 实验将 *N*、*w*、*R*、*SharedP* 和 *BackTrackMax* 分别设定为 7、4、0.25、0.2、0.5、9, 实验结果如图 2 所示. 从结果可以看出随使用资源概率的增加, 三种算法的调度成功率都开始降低. 调度算法在确定作业最早开始时间时需要考虑资源的使用状况, 而此时出现的资源冲突无疑推后了作业的开始时

间, 使得留给作业的剩余时间减少, 从而在某种程度上影响调度的成功率. 注意到, 当使用资源的概率较大时, 新算法能利用并行性保证作业可在时限之前完成, 但其他的两种算法不具备这样的特性, 因此调度成功率较新算法为低.

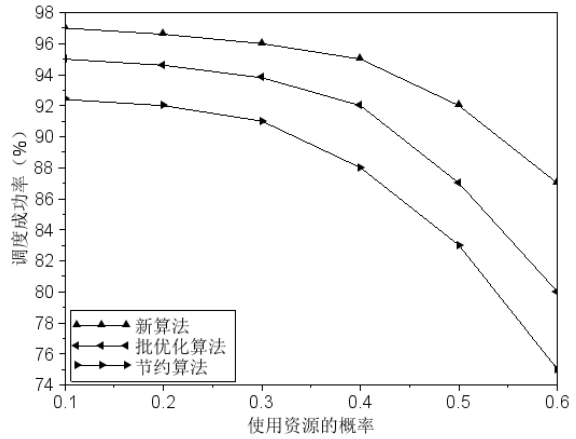

图 2 使用资源的概率对调度成功率的影响

图 3 描绘了最大回退次数 *BackTrackMax* 对作业集合调度成功率的影响. *N*、*w*、*R*、*UsedP* 和 *SharedP* 分别为 7、4、0.25、0.2 和 0.5. 可以看出最大回退次数对调度成功率的影响并没有想象中的那么大, 即随着系统所允许的最大回退次数的增加, 作业集合的调度成功率的提高变得缓慢. 这就从另一方面说明需要改变思路, 从其他方面寻找突破点. 本文提出的分批并行处理机就是这样的一种尝试. 从图中还能看出, 新算法的调度成功率优于批优化算法和节约算法.

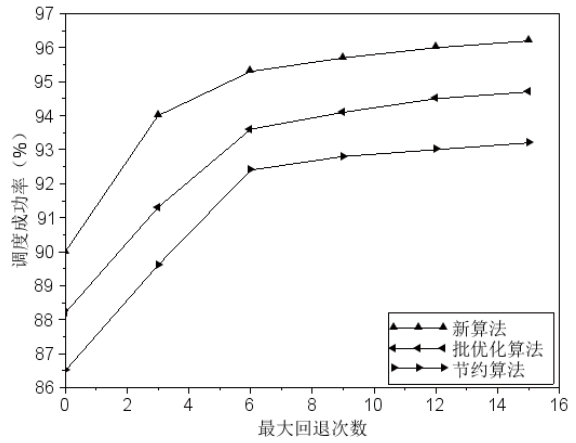

图 3 最大回退次数对调度成功率的影响

分批窗口大小 *N* 越大表明调度越有预见性. 另一方面, 如果 *N* 很大, 将会造成算法复杂度的上升. *w*、*R*、*UsedP* 、*SharedP* 和 *BackTrackMax* 分别是 4、0.25、0.2、0.5、9. 实验结果如图 4 所示. 当窗口大小增大时, 三个算法的调度成功率都有上升. 另外可看出, 新算法对分批窗口大小的变化最不敏感. 这就说明, 在维持较低成本的同时, 新算法能够提供更好的表现, 而这种特性是批优化算法和节约算法所不具备的.

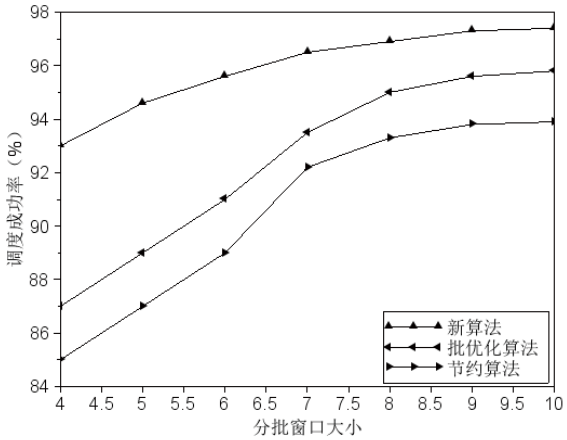

图 4 分批窗口对调度成功率的影响

5 总结与展望

本文提出了一种新的多处理机实时动态调度算法. 每次扩展局部调度时都选取一批作业, 然后综合考虑作业的属性以及系统的状况构造成本矩阵, 利用Kumar算法的变体找到最优解. 当存在作业不能满足时限时, 作业则被分成若干部分并行执行. 大量的仿真和实验表明新算法为非周期性任务集合提供了更高的调度成功率. 今后, 任务间约束关系更加泛化将是一个的关注点, 并且处理机利用率等性能度量也应引起足够重视.

参考文献

1 Liu JWS. Real-time systems. Upper Saddle River, New Jersey. Prentice Hall. 2000.

2 Dertouzos ML, Mok AK. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. on Software Engineering, 1989, 15(12): 1497–1506.

3 Ramamritham K, Stankovic JA, Shiah P. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Trans. on Parallel and Distributed Systems, 1990, 1(2): 184–194.

4 Manimaran G, Murthy CSR. An efficient dynamic scheduling

(下转第 163 页)

2.4 模型应用

基于普适计算的智能教室为用户提供高效、透明、移动的办公环境. 当教师进入到特定教室, 智能系统能根据用户行为提供服务, 并根据用户上下文信息进行权限调整.

如教师 A 在上课时间进入教师, 则自动为用户开启多媒体设备, 若教师 B 收教师 A 委托进入, 则需根据教师 B 的信任度、教师 A 委托授权的信任度评价值等, 对教师 B 进行模糊授权推理, 并计算该次访问信任度区间值, 若高于教室权限阈值, 则开启相应设备, 否则拒绝请求.

该模型有几个优点, 首先, 该授权推理过程简单, 计算复杂度小, 易在通用设备, 尤其是手持式设备上使用; 其次, 对象信任度值采用区间值表示, 而对上下文信息要求也是模糊的, 更符合普适计算环境; 最后, 该推理过程充分考虑的上下文信息的实时调整, 为每个评价值设置权重, 让整个模块更具有动态性.

3 总结

普适计算目前已经成为一个研究热点, 而且传统的分布式计算的安全机制并不完全适合于普适计算. 本文结合访问控制技术, 提出了基于信任度的动态模糊访问控制模型 TDFACM, 针对信任度描述模糊性的特点, 采用区间模糊理论对主体信任度进行评估, 建立了信任度的模糊计算模型与授权模型, 并提出了信任度衰减机制. 可以看出, 通过模糊评判, 模糊决策对用户进行授权的机制, 能更好的反应实际情况, 体现访问控制的公平性, 也更能保证高级别的用户可优先使用资源. 在今后的工作中, 将进一步完善模型, 如结合检测器对用户行为进行评估, 建立多级安全评估模型, 确保安全授权.

参考文献

1 Weiser M. The computer for the twenty-first century. Scientific American, 1991, 265(3): 94–104.

2 Sakamura K, Koshizuka N. The eron wide-area distributed-system architecture for Ecommerce. IEEE Micro, 2001, 21(6): 7–12.

3 Zhang D, Chen E, Shi YC, et al. A kind of smart space for remote real-time interactive learning based on pervasive computing model. Lecture Notes in Computer Science, 2003, 2783: 297–307.

4 郭亚军,洪帆,沈海波等.普适计算面临的安全挑战.计算机科学,2007,34(6):1–3,12.

5 Bertino E, Bonatti PA, Ferrari E. TRBAC: A temporal role-based access control model. ACM Trans. on Information and System Security, 2001, 4(3): 191–223.

6 Bertino F, Gatania B, Dam I. GEO-RBAC: a spatially aware RBAC. 10th ACM Symposium on Access Control Models and Technologies. Sweden. ACM. 2005. 29–37.

7 Toahchoodee M. Access Control Model for Pervasive Computing Environments. Fort Collins, USA. Cokorado State University. 2010.

8 吴茜,叶永升,李苑青.基于区间值加权模糊推理的访问控制模型.计算机应用研究,2012,29(10):3842–3845.

9 张海娟.普适计算环境下基于信任的模糊访问控制模型.计算机工程与应,2009,45(27):107–112.

(上接第 121 页)

algorithm for multiprocessor real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1998, 9(3): 312–319.

5 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.

6 李建国,陈松乔,鲁志辉.实时异构系统的动态分批优化调度算法.计算机学报,2006,29(6):976–984.

7 Kumar A. A modified method for solving the unbalanced assignment problems. Applied Mathematics and Computation, 2006, 176(1): 76–82.