新鲜度敏感的上下文推理实时调度算法

林 欣 李善平 杨朝晖
(浙江大学计算机学院 杭州 310027)

摘 要：由于普适计算中上下文具有时效性，上下文推理器必须支持推理任务的实时调度。针对上下文推理结果在一段时间内仍然保持“新鲜”的特点，本文提出推理结果重复利用效率及其计算公式。在此基础上提出一种新鲜度敏感的上下文推理实时调度算法 FRSA，以推理结果重复利用效率作为判断依据结合任务的 deadline 进行调度，其目标是在推理器负载较重时达到较高吞吐量。实验表明，在推理器负载重时，FRSA 的系统吞吐量比经典调度算法(SJF，EDF，LSF 和 FCFS)高出 10%–30%。

关键词：实时调度；上下文感知计算；上下文推理；普适计算；新鲜度
中图分类号：TP391 **文献标识码**：A **文章编号**：1009-5896(2009)05-1185-04

Freshness-Aware Real-Time Scheduling Algorithm for Context Reasoning

Lin Xin Li Shan-ping Yang Zhao-hui
(*College of computer science*, *Zhejiang University*, *Hangzhou* 310027, *China*)

Abstract: Due to the dynamic nature of contexts in pervasive computing, a context reasoner has to support real-time scheduling of reasoning jobs. Due to the fact that reasoning results remain fresh within a period of time, the concept of reasoning result reuse efficiency and its computation method are proposed. Then a Fresh-aware Real-time Scheduling Algorithm (FRSA) is proposed to promote the system throughput when the reasoner is overloaded, which schedules reasoning jobs according to their result reuse efficiencies and deadlines. The simulation demonstrates that when the reasoner is heavily overloaded, the throughput of FRSA is 10% to 30% better than those of classic scheduling algorithms SJF, EDF, LSF and FCFS.

Key words : Real-time scheduling; Context-aware computing; Context reasoning; Pervasive computing; Freshness

1 引言

上下文推理是上下文感知计算中的一个重要环节，所谓上下文推理，是指在普适计算和上下文感知计算中将应用程序无法识别、使用的低级上下文(low-level context)，通过聚合、冗余纠错、推理等手段，转换成应用程序可以直接识别的高级上下文(high-level context)[1,2]。上下文是动态变化的信息，如果应用程序不能及时获得最新的高级上下文，将影响普适计算中的用户体验。因此，应用程序对上下文的查询带有时间约束，上下文推理器必须是实时系统。

经典的实时调度算法(如EDF[3]、LSF[4]、SJF[5]等)不适用于上下文推理的调度，因为它们并没有考虑到上下文推理的以下两点特点：(1)一条上下文推理结果可以被当前或者将来的多个应用程序所用，因此可以将推理的结果放入缓存，供将来相同上下文的查询请求使用；(2)应用程序对上下文有新鲜度约束，即在之前t时段以内得到的上下文可以认为是准确的。

基于以上论述，本文借鉴文献[6]中提出的上下文新鲜度的约束，提出一种新鲜度感知的上下文推理调度算法(FRSA)，以提高响应请求数量为目标，兼顾上下文推理任务的完成时间约束和推理结果的重复利用效率。实验表明，当系统负载不断增加时，FRSA相对于传统实时调度算法的吞吐量优势越来越明显。

2008-04-28 收到，2008-07-21 改回
国家自然科学基金(60473052，60773180)和浙江省自然科学基金(Y106427)资助课题

2 问题描述与基本假设

推理器主要由 5 个部分组成：查询接收器、任务队列、缓存、查询记录和推理模块，其工作流程如下：首先，查询接收器负责接收应用程序发送的高级上下文查询，在查询记录中登记该查询以及它的元属性(包括发出时间、查询的高级上下文、发出查询的应用程序和新鲜度要求)，并将该高级上下文和缓存匹配，若缓存中的高级上下文是在查询的新鲜度约束范围内更新的，则认为缓存命中，直接返回给应用程序。若缓存没有命中，则将请求作为一个任务插入任务队列。然后，根据实时调度算法排出对列中最先执行的任务并交给推理模块执行。执行结果进入缓存，用于响应任务队列中以及将来关于该高级上下文的查询。本小节提出本文研究问题的描述和合理的基本假设：

(1)对于某一个高级上下文 hc ，推理器通过推理模块推

理出 hc 的时间记为 e(hc)，包括低级上下文从数据源传输到推理器的时间和推理所花费的时间，推理器预先已知每一个高级上下文的推理时间；

(2)应用程序发出的高级上下文查询 hcr 可以描述为一个 5 元组 $<\text{app},\text{hc},i,d,f>$，推理器中的任务与上下文查询是等价的。其中 app 代表发出 hcr 的应用程序，hc 是 hcr 中所要查询的高级上下文，i 代表 hcr 发出的时刻，d 代表应用程序提出的查询任务完成的期限(即 deadline)，过了该期限返回的结果没有意义。f 为 hcr 的新鲜度要求，即如果缓存中的高级上下文是在 $[i-f,i]$ 时段内得到的，即可认为结果正确，它描述了应用程序对高级上下文动态性的认知。通常情况下 f 总是小于 d 。用 hcr.app，hcr.hc，hcr.i，hcr.d，hcr.f 分别代表 5 元组中对应的元素；

(3)相同应用程序对相同高级上下文的所有查询遵循相同新鲜度要求，即对该高级上下文动态性的认知保持一致。将应用程序 app 查询高级上下文 hc 使用的新鲜度记为 f(app,hc)。

(4)假设缓存容量足够大，不需要缓存置换；

(5)假设调度花费的时间相对于任务推理时间忽略不计；

(6)推理调度属于非抢占式调度，减少为了推理任务切换所带来的开销；

(7)假设每个任务的重要性均等。

基于以上描述的模型和基本假设，调度算法的性能指标是吞吐量，即成功应答的查询个数与查询总个数的比值。

3 FRSA 算法

3.1 推理结果重复利用效率

FRSA 算法是一种兼顾上下文推理任务的时间约束和推理结果重复利用效率的实时调度算法。某个推理任务 hcr 的重复利用是指执行该任务的结果可以被其他多个关于 hcr.hc 的查询所利用。这种查询可以分为两个部分：(1)在任务队列中所有关于 hcr.hc 的任务，这些任务的集合记为 PeerReply(hcr)；(2)根据 hc 推理结果更新缓存，将来命中此缓存项的查询集合，记为 CacheHit(hcr)。记任务 hcr 的重复利用效率为 RE(hcr)，令：

$$\text{RE}(\text{hcr})=\frac{|\text{PeerReply}(\text{hcr})|+|\text{CacheHit}(\text{hcr})|}{e(\text{hcr.hc})} \tag{1}$$

即执行任务 hcr 获得的高级上下文被重复利用的个数除以执行该任务的时间。式(1)中的|PeerReply(hcr)|可以由任务队列获得，根据假设，推理器预先已知 e(hcr.hc)，而|CacheHit(hcr)|需要推理器对将来缓存命中的状况进行预判，只有当未来的高级上下文查询在新鲜度规定的时间范围内到来才可以认为缓存有效，因此，|CacheHit(hcr)|等于每一个应用程序在新鲜度范围内再次发送同一个高级上下文查询的个数总和。对于某个应用程序 app，记将来在 f(app,hcr.hc) 时段内发送关于 hcr.hc 的查询个数为 CH(hcr,app)，我们可以按照式(2)得到它的估计值：

$$\text{CH}^*(\text{hcr},\text{app})=\text{frequency}(\text{app},\text{hcr.hc})\cdot f(\text{app},\text{hcr.hc}) \tag{2}$$

其中 frequency(app,hcr.hc)等于应用程序 app 发送高级上下文 hcr.hc 查询的频率，它和 f(app,hcr.hc) 可以根据查询记录得到。由此，|CacheHit(hcr)|的估计值如式(3)：

$$\begin{aligned}|\text{CacheHit}(\text{hcr})|^*&=\sum_{\text{app}\in\text{APP\_SET}}\text{CH}^*(\text{hcr},\text{app})\\&=\sum_{\text{app}\in\text{APP\_SET}}\text{frequency}(\text{app},\text{hcr.hc})\\&\quad\cdot f(\text{app},\text{hcr.hc})\end{aligned} \tag{3}$$

综上所述，在以吞吐量为性能指标的上下文推理实时调度中，我们关注查询的重复利用效率，而它的估算需要首先获得应用程序的新鲜度约束。

3.2 算法描述

FRSA 的基本思想是当推理器不过载时，按照请求的 deadline 顺序，逐个调度完成推理任务；当推理器过载时，丢弃重复利用效率低的任务，直至推理器不过载。这样可以保证在不过载情况下，吞吐量逼近等于 EDF 算法，而过载时，保证完成重复利用效率高的任务，提高系统吞吐量。其中不过载的定义是推理器可以给出一种调度，使得任务队列中的所有任务按照该调度顺序执行，都不会发生超时。显然通过枚举所有调度顺序来判断是否过载复杂度过高，因此我们给出了一个启发式规则的实现：新任务插入队列时以 deadline 从先到后排列，这样保证在不过载的情况下，严格按照 deadline 先后顺序执行(即 EDF 调度)，且任务一旦进入队列，两两之间的先后顺序不变。从任务队列中选择下一个可执行任务的方法由图 1 描述，首先取队列头，称为目标任务(第 3 行中的 hcr)，判断任务本身是否能在其 deadline 之前完成，如果不能，则立刻丢弃(第 4-6 行)。然后考察目标任务的执行是否会导致任务队列中排在其后面但 RE 值比它高的任务超时，具体的判断办法是：考察目标任务及其后 RE 值比它高的任务组成一个的序列 highREList(第 8 行)，若执行 highREList 中从目标任务到 tmpJob 的所有任务，将会令 tmpJob 超时(注意，当有多个任务的 hc 相同时，执行时间只算一次)，则这些任务中至少有一个需要被丢弃。根据 FRSA 丢弃重复利用效率低的任务的思想，highREList 中目标任务的 RE 值最低，因此选择丢弃目标任务(第 9-13 行)。如果 highREList 中的所有任务都不会超时，我们称目标任务为可执行，并交给推理模块执行推理。

3.3 算法正确性证明

本小节要证明以下命题：若假设以后任务队列中不再加入新的任务，且每次执行的推理任务严格按照图 1 中的代码选取出，则不会出现执行 RE 低的任务而抛弃 RE 高的任务。因为推理器在选取下一个要执行的任务时无法预测将来应用程序会发出什么请求，只能通过现有队列中的任务做判断，所以命题中假设任务队列不再加入新任务是有意义的。以上的命题可以描述为：

```
1: Job nextJob(){
2:     While(job queue is not empty)
3:         hcr = head of job queue
4:         if(e(hcr.hc) + nowTime > hcr.d)
5:             remove hcr from job queue //查询 hcr 将会超时
6:             goto line 3
7:         end if
8:         highRElist = the jobs in the job queue whose RE
                        are higher than RE(hcr)
              //从任务队列中找出 RE 比 hcr 高的查询，按
                顺序组成队列 highRElist
9:         for each job tmpJob in highRElist
10:          sumTime= Σ_{∃j, j preceeds tmpJob in highRElist, hc=j. hc,} e(hc)
11:            if(sumTime+e(tmpJob)+nowTime>tmpJob.
                 deadlne)
                     //如果执行 hcr，将会使得比 RE 值比
                       它高的查询
                     //将来会超时
12:                  remove hcr from job queue
13:                  goto line 3
14:              end if
15:        end for
16:      return hcr
17:    end while
18: }
```

图 1 从任务队列中选择下一个可执行的任务

定理 1　记队列中目标任务为 r_0，将队列中 RE 比 r_0 的任务按顺序抽取出来组成序列，记为 $\mathrm{seq}(r_0)$，若 t_0 时刻 r_0 被判定可执行，t_0 时刻以后任务队列不再加入新任务，且后面的任务只有在被判定可执行时才能被执行，则 $\mathrm{seq}(r_0)$ 中的任务都会被执行。

证明　记任务队列中排在任务 r_x 后且RE值比 r_x 高的序列为 $\mathrm{seq}(r_x)$，序列中的第 y 项记作 $r_{x,y}$。对于任意下标 x，r_x 的执行时间为 e_x，deadline 为 d_x，在任务队列中 r_x 之前的一个任务执行完成或被抛弃的时间为 t_x (例如，$r_{0,j}$ 所对应的参数分别为 $e_{0,j}$, $d_{0,j}$ 和 $t_{0,j}$)。r_0 被判定为可执行，可以形式化描述为

对于满足 $1 \le l \le |\mathrm{seq}(r_0)|$ 的任意 l，都有

$$t_0 + e_0 + \sum_{j=1}^{l} e_{0,j} < d_{0,l} \tag{4}$$

要证明定理 1，先从证明引理 1 开始。

引理 1　对于满足 $1 \le i < l \le |\mathrm{seq}(r_0)|$ 的任意 i，l，都有

$$t_{0,i} + e_{0,i} + \sum_{j=i+1}^{l} e_{0,j} < d_{0,l} \tag{5}$$

归纳法证明：(1)令 $t_{0,0} = t_0$，$e_{0,0} = e_0$，则当 $i = 0$ 时，式(5)成立。

(2)假设满足 $0 \le i < l \le |\mathrm{seq}(r_0)|$ 时，$t_{0,i} + e_{0,i} + \sum_{j=i+1}^{l} e_{0,j} < d_{0,l}$ 成立，假设在任务队列中，$r_{0,i+1}$ 之前最后一个可执行的任务为 r_k。若 $r_k = r_{0,i}$，则根据可执行定义易得到 $t_{0,i+1} + e_{0,i+1} + \sum_{j=2}^{l} e_{0,j} < d_{0,l}$。若 $r_k \ne r_{0,i}$，由于 $r_{0,i}$ 和 $r_{0,i+1}$ 是 $\mathrm{seq}(r_0)$ 中相邻的两个任务，所以 $r_k \notin \mathrm{seq}(r_0)$，$\mathrm{RE}(r_0) \ge \mathrm{RE}(r_k)$，并推出 $\mathrm{seq}(r_0) \subseteq \mathrm{seq}(r_k)$。设 $r_{0,l}$ 在 $\mathrm{seq}(r_k)$ 序列中排在第 $z(l)$ 个，即 $r_{0,l} = r_{k,z(l)}$ ($l \in [i+1, |\mathrm{seq}(r_0)|]$)。

由 r_k 是 $r_{0,i+1}$ 之前最后一个可执行的任务可以得到：

(a)　$t_{0,i+1} = t_k + e_k$；

(b)　$t_k + e_k + \sum_{j=1}^{w} e_{k,j} < d_{0,w}$，其中 $1 \le w \le |\mathrm{seq}(r_k)|$。

结合　$\mathrm{seq}(r_0) \subseteq \mathrm{seq}(r_k)$ 可以推出：

$$t_{0,i+1} + e_{0,i+1} + \sum_{j=i+2}^{l} e_{0,j} = t_k + e_k + \sum_{j=1}^{l} e_{0,j} \le t_k + e_k + \sum_{j=1}^{z(l)} e_{k,j} < d_{k,z(l)} = d_{0,l}$$

也就是说，可以从 $t_{0,i} + e_{0,i} + \sum_{j=i+1}^{l} e_{0,j} < d_{0,l}$ 推出 $t_{0,i+1} + e_{0,i+1} + \sum_{j=i+2}^{l} e_{0,j} < d_{0,l}$，所以归纳法证明成立，引理 1 得证。

对于任意 i 满足 $1 \le i \le |\mathrm{seq}(r_0)|$，由 $\mathrm{RE}(r_{0,i}) > \mathrm{RE}(r_0)$ 可以得到 $\mathrm{seq}(r_{0,i}) \subseteq \mathrm{seq}(r_0)$，设 $\mathrm{seq}(r_{0,i})$ 中的第 l 项在 $\mathrm{RE}(r_0)$ 中为 $y(l)$ 项。结合引理 1 可以推出：

$$t_{0,i} + e_{0,i} + \sum_{j=1}^{l} e_{(0,i),j} \le t_{0,i} + e_{0,i} + \sum_{j=i+1}^{y(l)} e_{0,j} < d_{0,y(l)} = d_{(0,i),l} \tag{6}$$

其中 $e_{(0,i),j}$ 是指 $\mathrm{seq}(r_{0,i})$ 中第 j 项的执行时间，$d_{(0,i),l}$ 是指 $\mathrm{seq}(r_{0,i})$ 中第 l 项的 deadline。

由式(6)得到 $r_{0,i}$ 也是可执行的，定理 1 得证。

4　仿真实验

4.1　实验场景设置

本实验模拟的场景是基于文献[7]提供的智能会议室模型，在会议室中有多种上下文感知的应用程序共同工作。其中，某些程序为每位用户提供一份运行实例(如智能手机、智能 PDA)，应用程序数目会随用户的数量而动态改变，由此改变推理器的负载。和文献[7]不同，本实验主要关注上下文查询和上下文推理的实时性能，比较 FRSA 和经典实时调度算法的系统吞吐量(即成功应答查询的比率)，涉及的参数按照高斯分布随机产生，数学期望和取值范围由表 1 给出(取值范围以外的值被滤除)，默认的方差为 1。每组模拟实验的最先一分钟作为“预热”不纳入结果统计，以保证查询记录具有一定的有效性，每组实验模拟时间约 1000s。

4.2　系统负载对吞吐量的影响

本实验通过改变同时参与查询的应用程序个数来改变系统负载，当应用程序个数逐渐增加时，单位时间发出的上

表 1 实验各参数取值范围

参数	取值范围	数学期望
每个应用程序关注的高级上下文个数(n)	3–20 个	7 个
应用程序对每个上下文查询周期(p)	0.5–500s	2s
新鲜度约束(f)	0.5–60s	1.5s
完成时间约束(d)	0.1–50s	1s
查询任务执行时间(e)	0.001–2s	0.01s

下文查询个数也增加，从而增加系统负载。作为对比，本文运行 4 种经典实时调度算法作为对比，即：SJF(最短运行时间优先)，FCFS(最早来临优先)、LSF(最小空闲时间优先)和 EDF(最早完成时限优先)。图 2 描述了当系统负载改变时，各个调度算法吞吐量的改变情况。可以看出，在负载较轻时(应用程序个数为 1–3 个)，所有调度算法的吞吐量都很接近，当负载逐渐增加时，FRSA 相对于其他算法的优势就逐渐明显。在应用程序数量超过 15 个时，FRSA 的吞吐量比最接近于它的 EDF 高出 10%，比性能最差的 LSF 高出 27%，随着负载继续增加，这种差距在不断被拉大。为了更深入地剖析此差距产生的原因，我们还统计了每组实验中推理模块实际执行的查询比例和查询结果重复利用的比例，分别在图 3 和图 4 中描述，此二者的和即为系统吞吐量。从图 3 中可以看出，SJF 以最短执行时间为标准，的确可以执行更多的查询，但它总是偏向于丢弃执行时间长的查询任务，而执行时间短的任务将反复被执行，这样重复利用效率就很低(如图 4 所示)，总体的吞吐量不高。FCFS 只是简单地按照任务来临的顺序执行，不考虑任务的缓急和执行时间，当系统负载较重时，执行的任务数就大大降低，但由于它不偏向执行某一类任务，因此重复利用效率比 SJF 高。LSF 以剩余空闲时间作为调度的衡量指标，偏向于选择执行时间长的任务，所以 SJF 实际执行的查询比例也很低，同时，由于执行时间长的任务被反复执行，所以它的查询重复利用效率也较低，导致在负载较重时，整体吞吐量最低。从图 3 中可以看出，EDF 的实际执行查询的比例比 FRSA 略高一些，但由于 FRSA 每次调度都考虑查询重复利用效率，因此在图 4 中显示其重复利用的比例大大高于 EDF，吞吐量也较高。

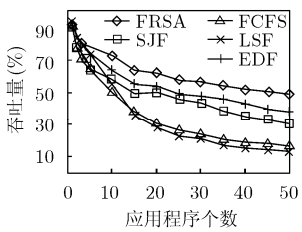

图 2 系统负载对各调度算法吞吐量的影响

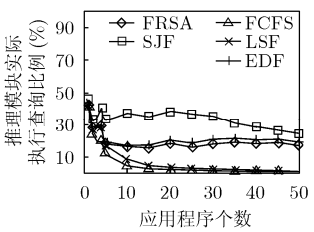

图 3 各算法实际执行查询任务的比例

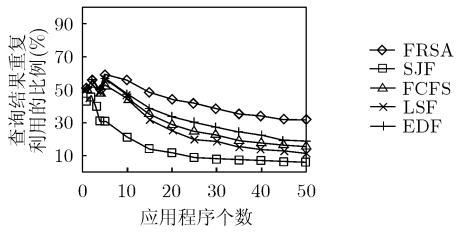

图 4 各算法查询结果重复利用的比例

5 结束语

本文针对上下文推理结果可以重复利用的特点，提出一种新鲜度敏感的上下文推理实时调度算法 FRSA，以推理结果重复利用效率作为判断依据结合任务的 deadline 进行调度，实验证明在负载较重时，FRSA 的系统吞吐量高于经典的实时调度算法。

下一步工作将针对具体的推理算法做实时调度优化，如，对于基于规则的层次化推理方法中可以利用缓存中间层次的推理结果来降低推理任务的执行时间，因此实时调度的模型更加复杂。另外，上下文感知计算的特性之一在于对上下文质量的处理，新鲜度是上下文质量的一种，其他的质量如分辨率、可信度等也可以作为调度时考虑的价值。

参 考 文 献

[1] 李蕊, 李仁发. 上下文感知计算及系统框架综述. 计算机研究与发展, 2007, 44(2): 269–276.
Li R and Li R F. A survey of context-aware computing and its system infrastructure. *Journal of Computing Research and Development*, 2007, 44(2): 269–276.

[2] Agostini A, Bettini C, and Riboni D. A performance evaluation of ontology-based context reasoning. Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), New York,USA, 19 - 23 March 2007: 3–8.

[3] Liu C L and Layland J. Scheduling algorithms for multiprogramming in real-time systems. *Journal of the ACM*, 1973, 20(1): 46–61.

[4] Dertouzos M L and Mok A K. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. on Software Engineering*, 1989, 15(12): 1497–1506.

[5] Silberschatz A, Galvin P B, and Gagne G. Operating System Concepts, Sixth Edition. John Wiley & Sons, Inc, 2002: 25–65.

[6] Han Q and VenKatasubramanian N. Timeliness-accuracy balanced collection of dynamic context data. *IEEE Trans. on Parallel and Distributed Systems*, 2007, 18(2): 158–171.

[7] Chen H L. An intelligent broker architecture for pervasive context-aware systems. [Ph.D. dissertation], Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 2004.

林 欣： 男，1981 年生，博士生，研究方向为普适计算、上下文感知计算.

李善平： 男，1963 年生，教授，博士生导师，主要研究方向为普适计算、系统集成.

杨朝晖： 男，1979 年生，博士生，研究方向为普适计算、上下文感知计算.