文章编号:1001 - 9081(2011)08 - 02075 - 04 doi:10.3724/SP.J.1087.2011.02075

新的普适计算环境下上下文缓存置换算法

王 斌[1],邹 文[1],盛津芳[1],孙 英[2]
(1. 中南大学 信息科学与工程学院,长沙 410083; 2. 中冶长天国际工程有限责任公司 自动化分院,长沙 410007)
(wb_csut@163.com; dfzwno1@yahoo.com.cn)

摘 要:由于普适计算环境高度动态的特点以及无线环境连接易中断、传输速度低等方面的约束,使得普适计算应用对于上下文的访问开销非常大。为此,首先给出一个上下文感知系统框架,然后提出了一种基于规则的上下文缓存置换算法——RCRA,算法根据上下文的被访问概率、上下文时效性及历史访问次数决定是否将其置换出缓存。当有新的上下文需要进入缓存时执行该算法,以保证缓存中的上下文最新且最有价值。实验表明,RCRA 不仅在命中率方面有较大提高,而且能够有效降低上下文访问的开销。算法应用于基于推理规则的上下文感知系统中,具有良好的可实用性。

关键词:上下文感知;上下文缓存;缓存置换算法;上下文价值;普适计算
中图分类号: TP311.52 **文献标志码**:A

New context caching replacement algorithm in ubiquitous computing

WANG Bin[1], ZOU Wen[1], SHENG Jin-fang[1], SUN Ying[2]
(1. *School of Information Science and Engineering, Central South University, Changsha Hunan* 410083, *China*;
2. *Automation Branch, Zhongye Changtian International Engineering Company Limited, Changsha Hunan* 410007, *China*)

Abstract: Due to the high dynamics of pervasive computing environment and the constraints of easy interruption and low transmission rate of wireless network, the overhead of context access is very huge. To solve these problems, a framework of context-aware system was proposed in this paper, and then a context caching replacement algorithm based on rules (RCRA) was introduced. The proposed algorithm determined whether to replace a context in the cache based on its access probability, timeliness and access history. When a new context was to get into the context cache, the algorithm was used to ensure that the latest and the most valuable context stayed in the cache. The experimental results show that the RCRA improves the hit rate and effectively reduces the overhead of context access. The RCRA is used in the rules-based context-aware system, and the algorithm has good utility.

Key words: context-aware; context cache; cache replacement algorithm; context value; ubiquitous computing

0 引言

在普适计算环境下,如何实现上下文信息的有效利用,实现上下文信息在应用程序、上下文中间件及传感器间的高效传播,是上下文感知计算的一个重要课题,也是一个难点[1]。这是因为:1)普适计算中计算实体具有动态性和持续变化性,同时受无线传输带宽的限制,使得应用对上下文的访问往往存在延迟,获取到的上下文已过时或包含错误;2)普适计算环境中上下文信息数量巨大,传输这些信息将占用宝贵的无线带宽,造成网络拥塞;3)无线网络连接的不稳定性有可能使有些上下文来源与应用程序间发生短暂或长时间的链接中断,直接影响到上下文感知在普适计算中的有效应用。

1 上下文感知系统

图 1 所示为本文提出的上下文感知系统框架,模型共分为 3 个层次:射频识别(Radio Frequency Identification, RFID)上下文感知服务层、RFID 上下文感知中间件层和底层通信环境。RFID 上下文感知服务层根据从 RFID 上下文感知中间件层获取到的消息作出相应处理,或者请求查询服务向 RFID 上下文感知中间件层提供摘要以获得高层的上下文信息。

底层通信环境包括各种 RFID 标签和读卡器及其他的传感器,它能够为 RFID 上下文感知中间件层提供各种原始的上下文信息。

RFID 上下文感知中间件层(以下简称中间件层)与底层通信环境进行通信以收集上下文信息,感知和捕获上下文信息和变化,为上下文感知的应用提供对上下文信息的访问(Search Services,pull 方式)或通告上下文发生的变化(Event Manager,push 方式)。中间件层同时负责管理上下文池,保持上下文池中的上下文最新最具价值。当对上下文池利用缓存置换算法清理过时和无效的上下文时,用户感兴趣的上下文的价值得到提高,在缓存池中保存更长的时间;中间层会主动与底层传感器通信更新这些上下文,若有更新,则及时通知应用程序,或者应用程序主动访问时能够直接从上下文缓存池获取到最新的上下文。

中间件层主要包括 2 大模块:上下文演化模块和上下文管理模块。上下文演化模块由上下文感知器、上下文解释器

收稿日期:2011 - 03 - 04;**修回日期**:2011 - 04 - 26。
基金项目:国家自然科学基金资助项目(60970039);湖南省自然科学基金资助项目(07JJ6124)。
作者简介:王斌(1973 -),男,山西大同人,副教授,博士,主要研究方向:软件工程; 邹文(1986 -),男,湖南娄底人,硕士研究生,主要研究方向:上下文感知计算; 盛津芳(1971 -),女,湖南长沙人,副教授,博士,主要研究方向:基于构件的软件工程; 孙英(1963 -),男,湖南长沙人,教授级高级工程师,硕士,主要研究方向:控制理论、控制工程。

和上下文推理器组成。经过上下文演化后,低层原始上下文被演化成应用所需的、格式统一的上下文信息,从而为各应用提供更好的服务。上下文管理模块负责上下文池、本体库及RF动态tag管理。上下文池管理负责维护上下文池中的上下文,保持上下文为最新状态。当有新的上下文需要进入缓存池时,执行上下文缓存置换算法,保证缓存池中的上下文的价值最大。RF动态tag管理模块负责更新标签memory,将更新了的tag值写回到标签中。

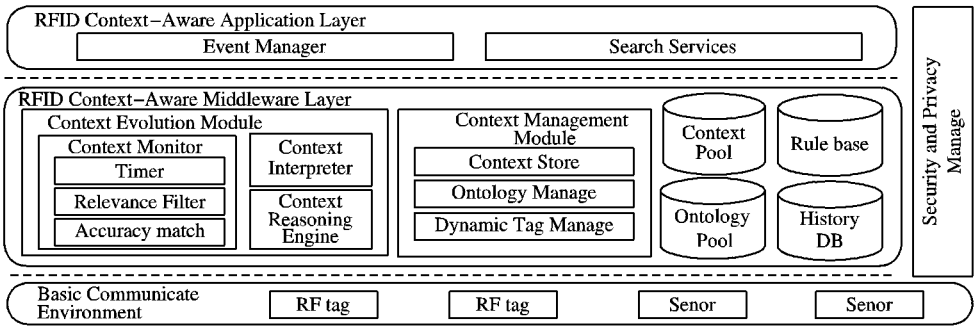

图1 上下文感知系统框架

上下文演化模块中,上下文感知器负责上下文信息和变化的感知和捕获。通过上下文感知器,中间件层能够提供一个抽象的接口,用户面对的是需要的上下文信息,而不必关心具体传感器的控制和信息的获取,实现应用与上下文处理的关注分离。本模块被用来捕获上下文信息发生的变化,并通知订阅模块。上下文推理器使用基于上下文本体和规则的推理模式,使用本体、规则及当前的上下文信息进行推理。

上下文管理块主要负责上下文池中上下文的管理,实现对上下文的保存及上下文的更新,以便于未来的使用。上下文池是针对上下文管理器的应用端的主要入口点。上下文管理块还负责对过时和冗余的信息进行管理。在陈旧上下文的清除中,通过判断上下文的价值,将"价值最低"的上下文信息清除。

2 上下文缓存

缓存置换算法的主要目的是当缓存池空间不够时,找出缓存价值最低的缓存数据将其替换,将缓存的数据规模维持在一个合理的数量,从而提高缓存的性能。在上下文感知计算环境下,当有新的上下文数据需要进入缓存,而缓存池中的上下文又达到一定数量已无法将新数据添加入缓存时,缓存置换算法需要选择缓存数据进行替换,使缓存中的数据最具有价值。传统的缓存置换算法有最近最少使用(Least Recently Used, LRU)算法、最少使用频率(Least Frequently Used, LFU)算法及先进先出(First In First Out, FIFO)算法。然而在普适计算环境中,上下文信息大小格式不一,各智能设备的传输速率差异较大,缓存失效造成的访问开销不同,这些因素在传统算法中都没有考虑在内,所以传统算法并不适合作为普适环境下的缓存置换算法。

Bdisk(Broadcast disks)是较早的无线环境缓存置换算法的项目之一,其中Acharya等人[2]提出了PIX(P Inverse X)缓存置换算法,算法以p/x值判断缓存池中数据是否应该被置换,其中p(Probability of access)是未来数据的访问概率,x(Frequency of broadcast)是数据的广播频率。Khanna等人[3]以PIX为基础提出了一种改进的算法(a paging algorithm G, Gray algorithm),综合考虑了缓存池中数据的访问历史和数据从服务器获取的延迟。而对于普适计算环境,PIX和Gray算法无法满足上下文信息大小不一致且频繁变动的要求。Xu等人[4-5]以两算法为基础,先后提出了SAIU(Stretch, Access-rate, Inverse Update-frequency)和MIN-SAUD(Minimum Stretch integrated with Access rates, Update frequencies, and cache validation Delay)算法,全面考虑了缓存数据的大小、更新率及数据失效带来的开销,因而算法相对客观和全面。然而,普适环境下的上下文具有持续变化性,由于上下文频繁更新,将上述算法应用于上下文缓存,将使缓存数据变得容易失效,缓存的命中率不高。由于普适计算环境的高度开放性,上下文使用者和提供者频繁地加入和退出,很难获得系统对上下文较长时间的访问历史,无法正确估计出上下文的访问概率。

在RFID上下文感知系统框架中,上下文池位于上下文感知中间件层,是针对上下文管理器的客户端应用的主要入口点,对系统中通过推理获取到的高级上下文进行缓存管理,其中缓存置换算法是实现缓存有效管理的关键性问题之一。

上下文缓存的概念由Anandarajah等人[6]提出,在其设计的上下文建模语言(Context Modeling Language, CML)的基础上,提出一种"智能"上下文缓存的设想,拟解决上下文在无线环境中传输可能遇到的断网问题。该设想从两个方面来判断上下文缓存的价值:上下文被访问的可能性(Possibility of Access, PA)和上下文的更新频率(Update Frequency, UF)。而上下文的价值通过PA/UF来衡量,PA/UF值越大,则该上下文的价值越高。但由于PA和UF的值是通过CML中上下文质量而获得,有些上下文的UF值无法从模型中获得,需要人工预先设置,从而无法达到普适计算环境及应用的一般性要求。

Olston等人[7]中提出了一种"Approximate Cache"的思想。所谓"Approximate Cache"是指缓存池并不存储每个缓存数据源精确的值,而只存储数据相近的范围。对于某些应用程序,对于数据的精度要求不高,只需要知道数据近似的范围即可。当新获取到的需要进入缓存的数据超过了这个范围,缓存则更新这些数据。这种思想主要关注的是如何把缓存数据的精度控制在一定的范围内,从而达到既满足了应用程序对数据精度的要求又可以降低通信带来的花费代价。文献[8]在文献[7]的基础上提出一种称为状态空间的推理建模方法,在此推理模型中,提出一种基于推理的上下文缓存置换算法(COntext Replacing Algorithm, CORA),其中主要包括低级上下文访问概率计算、低级上下文有效时间的判定和上下文一致性的保证。

基于上述研究,本文提出了基于规则的上下文缓存置换

算法(Rule-based Context caching Replacement Algorithm, RCRA),采用上下文的未来可能访问概率、上下文新鲜度及上下文的历史访问次数来判定上下文的价值,目标在于针对普适环境的以上特点,提高上下文缓存命中率。相对传统的缓存置换算法,RCRA 能够有效地提高缓存命中率,减少推理器和底层通信环境传感器的通信次数,从而降低无线传感器的能耗,减少无线网络的拥塞。

3　上下文缓存置换算法

文献[8]定义应用程序与高级上下文的相关函数 Cor_app_high,若某个应用程序 App_k 关注某条高级上下文 R_i,则 Cor_app_high 为1;否则 Cor_app_high 取值为0。在其计算上下文访问概率时,用 $Cor\_high\_sen(R_i,C_j)$ 表示某低级上下文 C_j 与高级上下文 R_i 的相关度,若 C_j 与 R_i 相关,则取 $Cor\_high\_sen(R_i,C_j)$ 值为1;否则 $Cor\_high\_sen(R_i,C_j)$ 取值为0。应用程序和低级上下文之间的相关函数用 $Cor\_app\_sen(App_k,C_j)$ 表示,如式(1)所示:

$$Cor\_app\_sen(App_k,C_j) = \sum_{i=1}^{N_{high}} Cor\_App\_high(App_k,R_i) \times Cor\_high\_sen(R_i,C_j) \quad (1)$$

其中:$Cor\_app\_high(App_k,R_i)$ 是应用程序 App_k 直接交给上下文中间件,$Cor\_high\_sen(R_i,C_j)$ 由推理器通过 R_i 的边缘函数获得。而所有应用程序的关注程度的总和则决定了低级上下文 C_j 被访问的概率,因此用式(2)表示某条上下文访问概率 G_j:

$$G_j = \sum_{k=1}^{N_{app}} W_k \times Cor\_app\_sen(App_k,C_j) \Big/ \sum_{j=1}^{N_{sen}} \sum_{k=1}^{N_{app}} W_k \times Cor\_app\_sen(App_k,C_j) \quad (2)$$

文献[8]给出了上下文缓存置换的一种比较合理的方法,但是却忽略了一个问题,在普适环境下的应用中,低级上下文的数量及种类往往是非常繁多的,而如果根据状态空间计算每条低级上下文的访问概率,会耗费有限的系统资源,可行性不高。RCRA 根据推理规则判断上下文的价值进而决定是否将该上下文置换掉。

在上下文感知服务系统中,高级上下文大都是通过低级上下文通过推理得出来的,如:作业人员“Zhang San”的“L_1:年龄是45岁”,当他的“L_2:血压收缩压大于120,舒张压大于80”时,且“L_3:脉博大于100次/min”时,通过规则推理认为“Zhang San”身体已处于“H_i:过度劳累”状态,即:“(L_1, L_2, L_3)→H_i”表示一条规则,L_1、L_2、L_3 表示三条低级上下文,H_i 表示由这三条低级上下文推理得出的高级上下文。本文将推理规则表示为“if…then…”的格式,即“左键→右键”的形式,其中 L_1、L_2、L_3 即为一条规则的“左键”,而 H_i 为规则的“右键”。因此由低级上下文 L_1、L_2、L_3 共同“产生”出了高级上下文 H_i,即只要低级上下文在某条规则的左键中出现了,则认为这条低级上下文对高级上下文 H_i 的产生“有贡献”,即与 H_i 有关。因此本文通过低级上下文在所有规则中出现的次数来判断这条低级上下文未来可能被访问的概率。对于规则的管理,本文将规则的左键和右键分开,并将其存储在数据库中,因而只要计算每条上下文在左键中出现的次数,即能断定这条上下文在规则中出现的次数。本文用 $Re\_Low\_High(L_i,R_j)$ 表示低级上下文 L_i 与高级上下文 R_j 之间的相关度,即 L_i 出现在用于推理 R_j 的规则里,则 $Re\_Low\_High(L_i,R_j) = 1$;否则 $Re\_Low\_High(L_i,R_j) = 0$,L_i 在规则中的出现次数越多,则认为其受关注度越高,未来可能被访问的概率也越大。L_i 的未来被访问概率 G_i 计算公式如式(3)所示:

$$G_i = \sum_{j=1}^{N_{high}} Re\_Low\_High(L_i,R_j)/N_{high} \quad (3)$$

在 RCRA 中,V_i 用于表示一条低级上下文 i 的价值,V_i 计算方法如式(4):

$$V_i = C_1/(T_{now} - T_i) + C_2 \times P_i + C_3 \times G_i \quad (4)$$

V_i 的值由 P_i、G_i、T_i 共同确定,其中:P_i 表示一条上下文已被访问的次数,可以直接获得;G_i 表示一条低级上下文未来被访问概率;T_i 表示一条上下文的产生时间;T_{now} 代表现在时刻;($T_{now} - T_i$)为上下文的新鲜度;C_1、C_2、C_3 分别为 T_i、P_i、G_i 的加权因子。由式(4)计算所得的 V_i 值可知,在缓存中,上下文的访问次数越小,V_i 值也就越小;上下文产生的时间越长,V_i 值也就越小;上下文未来被访问概率 G_i 越小,则 V_i 值也就越小。因此,每次替换数据对象时都替换 V 值最小的上下文。由以上分析可以看出,低级上下文价值 V_i 判断算法如下所示:

```
Algorithm RCRA (inContext, C)
Input: the contexts (inContext) that need to save to cache, the
       weighting factors C = C1, C2, C3 of T, P and the future access
       probability (G) of inContext;
Output: True/False;
if RemainSize(cacheContext) > SIZE_inContext or cacheContext ==
NULL then
IntoCache (inContext);
end if
for i = 1 to SIZE_cacheContext do
  G_i ← Compute(cacheContext_i);
  V_i ← C1/(T_i - T_now) + C2 * P_i + C3 * G_i
end for
while ((cacheSize - contextSize) < SIZE_inContext) do
   Remove (cacheContext(Find-Min(V_i)));
end while
IntoCache (inContext);
return True
```

4　实验分析

由于上下文缓存的主要目标是要减少上下文监控模块与底层通信环境传感器的通信次数,减少无线网络中的传输量,因此实验的主要性能指标是缓存的命中率,并以 LRU 为参照算法。

实验模拟 k 个应用程序向上下文池查询 h 个高级上下文,上下文池通过上下文推理模块向 m 个传感器查询低级上下文的场景。

本文假设每个高级上下文的访问频率遵循参数为 θ 的 Zipf 分布,Zipf 分布中 θ 越大说明对高级上下文的访问集中在少部分的上下文,当 θ 趋于0时表明对高级上下文的访问趋于均匀分布。实验分别采用 RCRA 和 LRU 两种算法对低级上下文进行缓存,缓存容量为 C,用于推理的规则数目为 R,低级上下文的数目为 L,更新频率为 U,高级上下文数目为 H,访问频率为 F。实验参数设置如表1所示。

图2是 Zipf 分布偏量从0到1对缓存命中率的影响,其他参数设为默认值不变,见表1。当 θ 等于0时,应用程序访问高

级上下文的概率为均匀分布，即随机访问，所以 RCRA 和 LRU 的命中率都很低。随着 θ 的不断增大，应用程序访问的大部分上下文集中在越少部分的高级上下文中，因此缓存的重复访问频率也随之加大，被重复访问的高级上下文价值 V 随着历史访问次数的增多也变得越大，因而被保留在缓存中不会被替换，故 RCRA 的命中率随之有较大提升。LRU 算法的性能也有所提升，但 LRU 由于认为低级上下文只要有更新就会失效，导致性能大幅度低于 RCRA。

表 1 实验参数表

参数	默认值	意义
C	12%	缓存所占低级上下文个数的百分比
K	10	访问上下文程序个数
R	3 000	推理规则数目
L	1 200	低级上下文条数
H	100	高级上下文条数
U/F	3∶1	低级上下文更新频率/高级上下文访问频率
θ	0.9	Zipf 分布的偏量参数
C_1, C_2, C_3	0.3,0.2,0.5	上下文产生时间、被访问次数、被访问概率加权因子

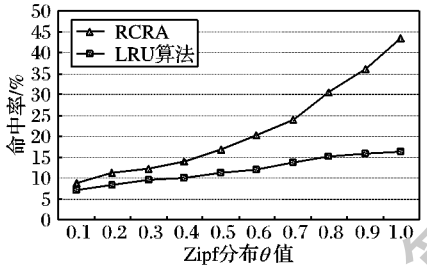

图 2 Zipf 分布 θ 值对算法命中率的影响

图 3 是缓存所占低级上下文的比例从 1% 到 10% 对缓存命中率的影响，其他参数设为默认值不变。取 θ 值为 0.9，即对上下文的访问集中在少数高级上下文。随着缓存大小的变大，RCRA 和 LRU 算法的命中率都得到提高。RCRA 的命中率稳步上升，LRU 算法的命中率也有所提升，但是明显低于 RCRA 的命中率，LRU 算法对缓存的大小并不敏感，这是因为 LRU 认为上下文一旦更新就会失效，而实际上很多上下文的访问次数会很多，而且上下文也频繁更新，LRU 对于这种情况不能很好地处理，因而性能远比不上 RCRA。

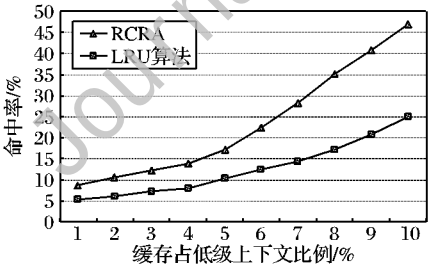

图 3 缓存大小对算法命中率的影响

图 4 是更新速率与查询速率比从 1∶10 到 10∶1 的缓存命中率变化。随着更新频率的提高，RCRA 的性能略微有些降低，而 LRU 的性能下降很大，特别是在更新速率与查询速率比大于 1 时，LRU 的命中率急剧下降直至接近为 0。这是由于当更新速率大于查询速率时，缓存中的数据被频繁重复查询，因而系统也会频繁与传感器端通信获取新的上下文，缓存中大量的上下文都处于生命周期的更新态，LRU 访问时，缓存池的数据已经失效，而 RCRA 认为这些上下文只是更新了，仍然认为有效。

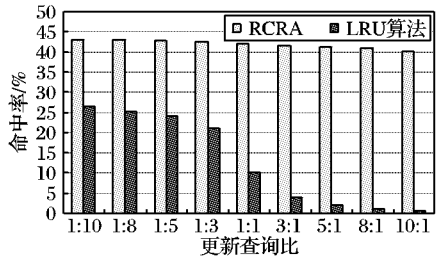

图 4 更新频率对算法命中率的影响

跟 CORA 相比，在缓存大小对算法命中影响方面，RCRA 命中率要高于 CORA，而且 RCRA 的性能要相对稳定些，在 Zipf 分布 θ 值对算法的影响方面，RCRA 的命中率也略微高于 CORA。另外由于 RCRA 应用于基于推理规则的上下文感知系统，RCRA 具有很好的性能，尤其对于实际已搭建好知识库的上下文感知系统来说，RCRA 能够充分发挥其作用，因此较之 CORA，RCRA 具有更好的实用性和可实现性。

5 结语

文章首先给出了 RFID 上下文服务模型框架，阐述了框架中各个模块的作用。分析了上下文缓存的必要性以及上下文缓存的研究背景，指出现有无线环境中的数据缓存置换算法不太适用于普适计算环境上下文缓存。为此，提出一种上下文缓存置换算法——RCRA，重点介绍了低级上下文价值计算方法。最后通过模拟实验，将 RCRA 和 LRU 算法进行对比，从 3 个方面验证了 RCRA 的性能比 LRU 算法有很大提高。

参考文献：

[1] SAFA H, ARTAIL H, NAHHAS M. A cache invalidation strategy for mobile networks [J]. Journal of Network and Computer Applications, 2010, 22(2): 168 - 182.

[2] ACHARYA S, ALONSO R, FRANKLIN M, *et al*. Broadcast disks: Data management for asymmetric communications environments [C]// SIGMOD'95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1995: 199 - 210.

[3] KHANNA S, LIBERATORE V. On broadcast disk paging [J]. Journal of Computing, 2000, 29(5): 1683 - 1702.

[4] XU J, TANG X, LEE D L. Performance analysis of location-dependent cache invalidation schemes for mobile environment [J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(2): 474 - 488.

[5] XU J, HU Q, LEE W C. Performance evaluation of an optimal cache replacement policy for wireless data dissemination [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(1): 125 - 139.

[6] ANANDARAJAH M, ROBINSON R. Caching context information in pervasive systems [C]// Proceedings of the 3rd International Middleware Doctoral Symposium. Melbourne, Australia: [s. n.], 2006: 20 - 27.

[7] OLSTON C, LOO B T, WIDOM J. Adaptive precision setting for cached approximate values [J]. ACM SIGMOD Record, 2001, 30(2): 355 - 366.

[8] 林欣，李善平，杨朝晖，等. 普适环境中面向推理的上下文缓存置换算法[J]. 计算机研究与发展，2009，46(4)：549 - 557.