可重写循环滑动窗口:面向高效的在线数据流处理*)

李俊奎　王元珍
(华中科技大学数据库与多媒体研究所　武汉 430074)

摘　要　滑动窗口是在线数据流处理中的重要技术和基础设施。针对当前基于向量模型的滑动窗口存在滑动过程中需要移动过多数据,而导致效率不高的问题,本文提出一种可重写循环的滑动窗口技术。该技术在滑动过程中不移动数据,而是采用重写的方式来完成数据更新,并且它能够与当前滑动窗口无缝集成。理论分析和实验对比表明,该技术有显著的效率提升,能够高效地应用于实际的数据流处理。

关键词　在线数据流,滑动窗口,可重写,循环

Rewritable Circular Sliding Window: Towards Effective On-line Data Stream Processing

LI Jun-Kui　WANG Yuan-Zhen
(Research Institute of Database & Multimedia, Huazhong University of Science & Technology, Wuhan 430074)

Abstract　Sliding window is one of the important techniques as well as fundamental infrastructures in on-line data stream processing. The problem with current sliding window implementation is that the vector model needs to move data in the sliding window as the window slides, which has poor performance. A rewritable circular technique is proposed, which adopts a rewriting step to update the data in the window and allows no data movement during the process of sliding. The method can be integrated seamlessly to current sliding window implementation. The theoretical analysis as well as comparative experiments show, the method earns great promotion in light of efficiency, and can be effective in real data stream processing.

Keywords　On-line data stream, Sliding window, Rewritable, Circular

1　引言

数据流(Data Stream)是一串实时、连续和有序的数据序列。数据流在实际生活中广泛存在,典型应用[5]有互联网流量、传感器网络数据、电话记录、大气温度数据以及大量实时的商业数据等,当前对数据流的处理已成为一个研究热点[2]。

处理数据流面临新的挑战,这些挑战可以简单概括为"在线、量大、高速、未知、一遍"。

在线:数据在线到达而非已静态存储;

量大:数据流的瞬时数据量巨大;

高速:数据随着流的推进而快速变化;

未知:数据流的数据流速和大小未知;

一遍:一旦流中的数据被处理过,将被丢弃或被离线存储,再次处理则相当困难甚至几乎不现实。

尽管流数据可以看作是无限的,但是实际的计算都是在流数据一个较小的子集上进行,即存在一个窗口的概念[6]。窗口在流数据上不断滑动,窗口内的数据则不断得到处理。目前典型的滑动窗口实现都基于向量模型[1],这种模型的一个问题是随着窗口的滑动,需要移动窗口内的数据,把新数据移入窗口,旧数据移出窗口。

本文则提出一种可重写循环的滑动窗口方法,该方法采用对窗口数据的重写完成数据更新,在窗口滑动的过程中并不需要移动数据,从而提高了系统的处理效率。另外该方法可以与当前的滑动窗口实现无缝集成,应用该方法不会影响到已实现的流数据处理系统上层应用。

本文其余部分如下组织:第2节讨论当前基于向量模型的滑动窗口实现;第3节提出可重写循环的滑动窗口技术;实验结果以及实验分析在第4节给出;最后总结全文,并指出未来的进一步工作。

2　基于向量模型的滑动窗口

2.1　相关定义

定义1 数据流　是一串按照时间顺序无限到达的元组 $T=t_1,t_2,\cdots$,其中 $t_i(i\geqslant 1)$ 是数据流中标识(采用显式时间戳或隐式到达时间点标识)为 i 处的数据。

为简便处理,本文统一采用隐式到达时间点作为数据流的数据标识,但使用显式时间戳并不影响文中讨论。

定义2 滑动窗口　在数据流上滑动一个大小为 $w(w>0)$ 的窗口,窗口内数据 $S=s_0,s_1,\cdots,s_{w-1}$ 构成流数据的一个瞬时抽样,w 称为窗口宽度。

定义3 格局　滑动窗口内数据的一次状态/组织形式称为一个格局,位置为 $i(0\leqslant i\leqslant w-1)$ 的数据格局可以用三元组 $P(i)=\langle length,head,i\rangle$ 表示,其中:

length 为滑动窗口内数据量;

head 为滑动窗口末端标记;

i 为滑动窗口内的位置下标。

2.2　基于向量模型的滑动窗口技术

(1)基本思想

*)国家发展与改革委员会"安全智能数据整合平台开发及产业化"项目(项目编号[2005]538号)。**李俊奎**　博士研究生,主要研究方向:数据挖掘、机器学习;**王元珍**　教授,博士生导师,主要研究方向:现代数据库理论与实现技术、数据挖掘中间件技术。

当前数据流上的滑动窗口都基于向量模型实现，该向量模型将滑动窗口建模为一个向量。随着窗口的移动，靠近向量末端的数据向前移，覆盖前面的数据，新的数据则加入到向量末端，从而完成数据更新。

(2)形式化描述

该模型可以形式化表示为：

$VectorSW=\langle w,length,head,f\rangle$，

- w 为滑动窗口宽度；
- $length$ 为当前窗口的数据量；
- $head$ 为滑动窗口数据末端的标记，新数据放置在该位置；
- f 为滑动窗口的格局变换函数：$f: P\rightarrow P'$，它决定了在新数据到来时滑动窗口中已存在数据的格局变化。在向量模型中，f 的定义如表 1 所示。

表 1 向量模型滑动窗口格局变换函数

(nil、new 分别表示不记录格局和新格局)

$f(\langle length,head,i\rangle \mid length<w)$	
$0\leqslant i<head$	$\langle length+1,head+1,i\rangle$
$i=head$	new
$head<i<w$	nil
$f(\langle length,head,i\rangle \mid length=w)$	
$i<>head$	$\langle length,head,i+1\rangle$
$i=head$	new

(3)模型分析

从表 1 可以看出，基于向量模型的滑动窗口技术分为两个阶段：窗口未满和窗口已满阶段。

在窗口未满阶段，随着窗口的滑动，窗口不移出数据，新数据进入窗口的 $head$ 位置，同时更新 $length$ 和 $head$。

在窗口已满阶段，随着窗口的滑动，窗口内的第 1 个数据被移出，其它数据则前移，覆盖前序数据，新数据进入窗口的末端位置($w-1$)。此时 $length$ 值固定为 w，$head$ 值则固定为 $w-1$，表示新数据都在窗口末端进入窗口。

图 1 显示了基于向量模型的滑动窗口格局转换过程。

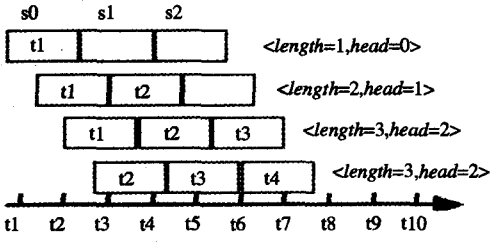

图 1 向量模型的滑动窗口格局转换示意图

3 可重写循环的滑动窗口

(1)基本思想

基于向量模型的滑动窗口方法中，在窗口已满阶段，新数据进入窗口内将导致其他已在窗口内的数据发生相应移动，但是此时可以把要移入的数据覆盖(重写)要移出的数据，并且提供一套机制来维护窗口数据的逻辑格局状态，从而在维护逻辑一致性的同时，又避免窗口内数据的移动。此即为可重写循环滑动窗口模型的思想来源。

(2)形式化描述

该模型可以形式化表示为

$CircularSW=\langle w,length,head,f\rangle$

- $w,length,head$ 含义与向量模型相同；
- $f: P\rightarrow P'$ 为格局变换函数，其定义如表 2 所示。

表 2 可重写循环滑动窗口格局变换函数

(nil、new 分别表示不记录格局和新格局)

$f(\langle length,head,i\rangle \mid length<w)$	
$0\leqslant i<head$	$\langle length+1,head+1,i\rangle$
$i=head$	new
$head<i<w$	nil
$f(\langle length,head,i\rangle \mid length=w)$	
$i<>head$	$\langle length,(head+1)\%w,i\rangle$
$i=head$	new

(3)模型分析

从表 2 可以看出，在滑动窗口未满阶段，可重写循环模型与向量模型相同，新数据直接填充窗口。在窗口已满阶段，随着窗口的滑动，与向量模型不同的是，可重写循环模型直接计算出即将被移出数据的位置(即$((head+1)\%w)$)，新数据则放置到该位置，覆盖窗口中该处的数据，同时修改 $head$ 值，指示窗口中数据的末端。

图 2 显示了可重写循环的滑动窗口格局转换过程。该模型底层可以采用类似于循环队列的方式实现[7]，但与循环队列不同的是，在窗口满时需要对窗口数据进行重写。

(4)与现有滑动窗口集成

由于滑动窗口是数据流处理的一种重要基础性技术，现有数据流处理系统大多都已经实现了滑动窗口，因此有必要将可重写循环的滑动窗口与现有滑动窗口集成，从而对数据流处理系统上层应用不构成大影响。

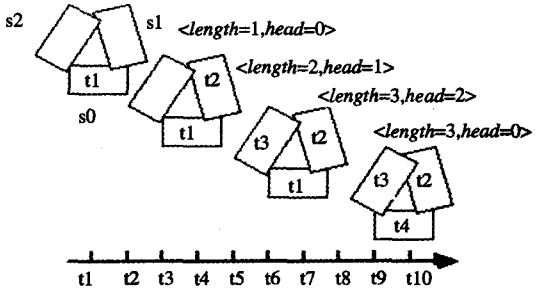

图 2 可重写循环滑动窗口格局转换示意图

本文以向量模型到可重写循环模型的转换来讨论二者集成。首先是二者转换的前提，这由定理 1 来保证。

引理 1 在相同输入后的每一时刻，基于向量模型和可重写循环模型的滑动窗口中含有相同的数据。

证明：根据窗口状态分情形证明。

(1)窗口未满。由于两种模型在窗口未满时动作相同，所以结论成立；

(2)窗口已满。在窗口刚满时，两种模型窗口中的数据具有相同的格局状态，记窗口宽度 $w(w>0)$，当前的窗口数据排列为$\langle d_0,d_1,\cdots,d_{w-1}\rangle$，此时 $head$ 值为 $w-1$。当新数据(设为 d)到来时，向量模型中窗口数据依次前移，第 1 个数据则被移出窗口，窗口的数据排列变化为：

$\langle d_1,d_1,\cdots,d_{w-1}\rangle\rightarrow\langle d,d_1,\cdots,d_{w-1},d\rangle$，

可重写循环模型中窗口数据不移动，而是改变 $head$($(head+1)\%w$)，数据排列变化为：

$\langle d_0, d_1, \cdots, d_{w-1}\rangle \rightarrow \langle d, d_1, \cdots, d_{u-1}\rangle$，

随后移出的数据顺序依次为 $d_1, d_2, \cdots$。

由此，两个模型中都保证了每次移出的数据都是出现在窗口内最久的数据，随着窗口的继续滑动，模型中的数据始终相同。

定理 1(对外等价性定理)　在相同输入后的每一时刻，基于向量模型和可重写循环模型的滑动窗口中的数据可以通过映射访问得到相同的结果。

证明：构造访问映射函数：$\mu: S \rightarrow C$，其中 S 为向量模型的地址下标集合，C 为可重写循环模型的地址下标(相对于 *head* 位置)，给定窗口宽度 w，$S=\{0,1,\cdots,w-1\}$，$C=\{head,(head+1)\%w,\cdots,(head+w-1)\%w\}$。

$$\mu(i)=(head+i)\%w, i\in S \qquad (1)$$

从引理 1 的证明过程易知，通过 μ 可以将向量模型中的数据映射到可重写循环窗口中的相同数据。证毕。

定理 1 不仅保证了向量模型和可重写循环模型的对外等价性，而且为实现从向量模型到可重写循环窗口的转换提供了思路。

由于可重写循环窗口在每次移入新数据时，采用移动 *head* 的方法，而定理 1 通过构造访问映射函数 μ 式(1))说明可以在向量模型中通过计算相应的移入下标来实现可重写循环窗口，从而完成向量模型到可重写循环窗口的转换。图 3 中给出了转换的示意图。新数据到来后，不写入窗口末端，而是写入将要移出窗口的数据位置。

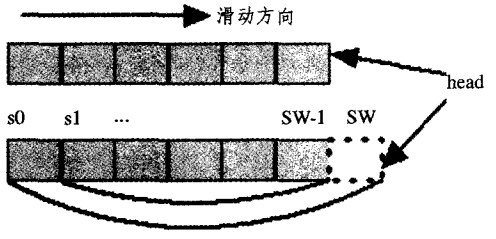

图 3　向量模型滑动窗口到可重写循环窗口的转换示意图

(5)效率提升分析

与基于向量模型的滑动窗口相比，可重写循环窗口能够提升数据流处理系统的效率，主要体现在以下方面：

· 同样的存储空间

可重写循环窗口没有增加窗口宽度，而是在原有窗口宽度的前提下，维护一致的窗口格局状态。这保证了可重写循环模型的改进不是以增加存储空间为代价的，至少可以取得与原滑动窗口一样的外部性能；

· 不需要移动数据

滑动窗口宽度相对于数据流数据量而言是相当小的，因此在短时间内滑动窗口即被填充满。在窗口已满阶段，基于向量模型的滑动窗口中的数据将依次前移，覆盖原位置数据，新数据则覆盖处于窗口末端的数据。而可重写循环滑动窗口采取的是覆盖即将移出窗口的数据的策略，新数据移入窗口不需要移动窗口内的数据。对于宽度为 w 的窗口，每次新数据到来，可重写循环滑动窗口可以节省 $w-1$ 次数据平移的开销。考虑到数据流处理的长期性，这种效率提升是巨大的；

· 允许更细粒度的并发控制

滑动窗口在数据流处理系统中充当数据提取手段和数据采样的工具，因此对于数据流而言，滑动窗口一直在写数据，对于数据流处理系统上层应用而言，滑动窗口则一直在读数据，在这种情形下，需要对滑动窗口进行并发控制。现假定利用加锁的方式来实现窗口的并发控制，由于向量模型的滑动窗口需要移动数据，在新数据移入完成前，整个滑动窗口的空间都处于锁定状态；而在可重写循环滑动窗口方案中，可以只对 head 处数据加锁，在新数据到来时，窗口内的其他数据可以为上层应用访问。

4　实验研究

4.1　实验准备

这一节对可重写循环滑动窗口模型的实际效果进行实验研究。可重写循环滑动窗口模型与基于向量模型的滑动窗口的一个最大的不同是，在不改变存储空间的情况下，可重写循环滑动窗口中的数据不会随着窗口的滑动而移动，而是通过重写实现新旧数据的更替，在实现对外逻辑一致性的同时，又能提高效率。

使用 C++分别实现了基于向量模型和可重写循环的滑动窗口，实验环境是：Microsoft Windows® 2000 Server，40G 硬盘，256M 内存，PIII-866CPU，g++(mingw special)3.2.3 编译环境。

实验分别在人工产生和实际的数据流数据上进行。人工产生的数据流来源于一个能够产生随机数的发生器程序。实际的数据流则来源于一台实际的信号发生器，该信号发生器能够以一定的频率产生实际的流数据，实验中为 100bps。

为表示滑动窗口的性能对上层应用的影响，实现两种时间序列数据的上层表示方法，分别是 PAA(Piecewise Aggregate Approximation)和 APCA(Adaptive Piecewise Constant Approximation)来测试滑动窗口的性能。具体有关时间序列表示方法 PAA 和 APCA 的原理请分别参见文[4]和文[3]。

4.2　实验结果及分析

(1)滑动时间

实验 1：在等量数据上未加上层应用的窗口滑动，对窗口不采用并发控制。实验主要考察窗口对于数据流的快速响应能力。实验结果如图 4 和图 5 所示。实验结果表明，可重写循环窗口的响应时间比向量模型的时间短，而且随着窗口大小的增大，两种模型的滑动时间都减少。窗口越大，单位时间内滑过的数据较多，在数据量一定的情况下，滑动时间相应减少，但可重写循环窗口由于不需要频繁移动数据，所以进一步减少了滑动的时间开销。

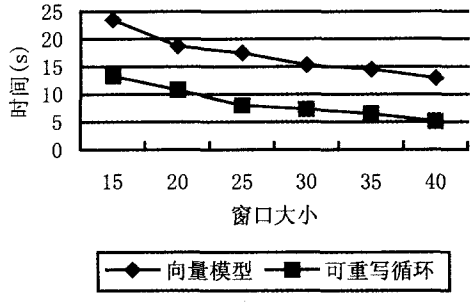

图 4　在人工产生的数据上滑动时间比较(未并发控制)

实验 2：在等量数据上运行 PAA 应用，对窗口采用加锁的并发控制。实验主要考察窗口大小对滑动时间的影响，结果如图 6 和图 7 所示。实验结果验证了可重写循环窗口的效率提升，有趣的是，实验中向量模型随着窗口的变大，滑动时间发生起伏。向量模型窗口滑动过程中需要对整个窗口锁

定，窗口越大，锁定粒度越大，阻碍了窗口的滑动速度。

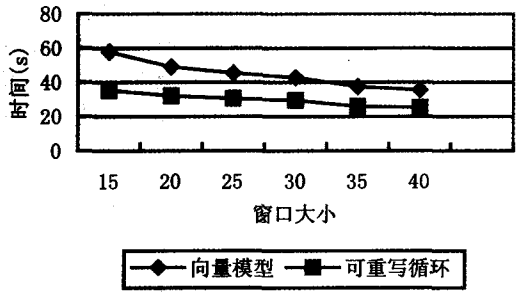

图 5　在实际数据上滑动时间比较(未并发控制)

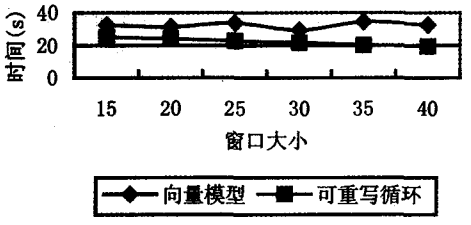

图 6　在人工产生的数据上处理 PAA 滑动时间比较(用加锁方式并发控制)

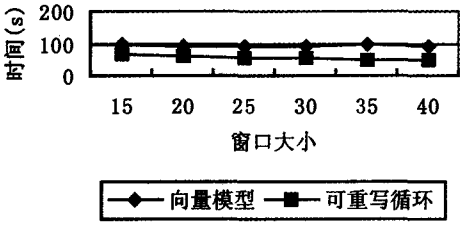

图 7　在实际数据上处理 PAA 滑动时间比较(用加锁方式并发控制)

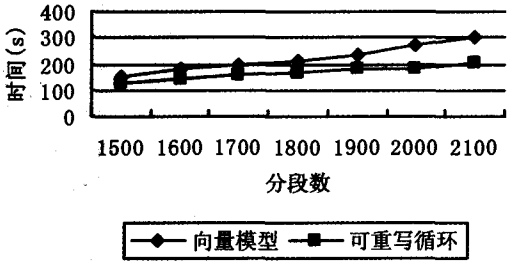

图 8　在人工产生的数据上处理 APCA 滑动时间比较(用加锁方式并发控制，δ=15.0)

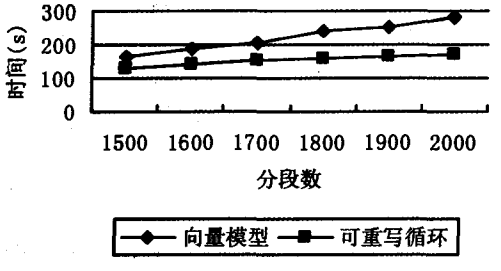

图 9　在实际数据上处理 APCA 滑动时间比较(用加锁方式并发控制，δ=15.0)

实验 3：运行 APCA 应用产生等量的数据分段，对窗口采用加锁的并发控制。实验主要考察数据量对于滑动时间的影响，实验结果如图 8 和 9 所示(为窗口数据的累积误差)。可重写循环窗口相比向量模型滑动时间更小，进一步验证了前面的分析结论。

(2)内存使用量

实验中还同时记录了处理 PAA 和 APCA 时的内存使用量，结果分别如图 10～13 所示。结果表明在相同条件下，向量模型和可重写循环模型的内存使用量没有大的差异，可重写循环模型并没有显著增加内存量。这与前面的分析相符。

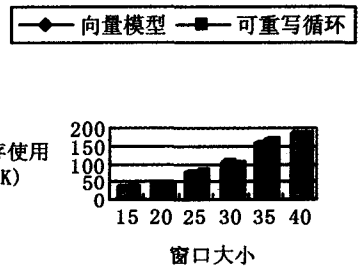

图 10　在人工产生的数据上处理 PAA 内存使用量比较(用加锁方式并发控制)

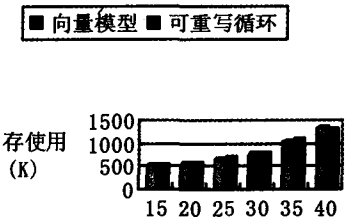

图 11　在实际数据上处理 PAA 内存使用量比较(用加锁方式并发控制)

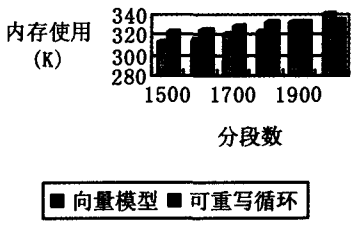

图 12　在人工产生的数据上处理 APCA 内存使用量比较(用加锁方式并发控制，δ=15.0)

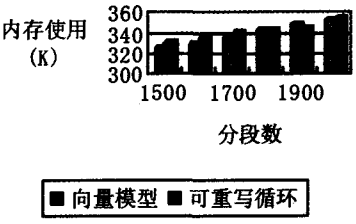

图 13　在实际数据上处理 APCA 内存使用量比较(用加锁方式并发控制，δ=15.0)

结论　本文提出了可重写循环滑动窗口模型，目的是避免向量模型在窗口滑动时移动数据，而是采用循环重写窗口中数据完成数据的更新。文中对两个模型进行了形式化描述，为了实现已实现的数据流处理系统上层应用不构成影响，对从向量模型到可重写循环模型的转换给出了方案。通过理

论和大量实验表明，可重写循环滑动窗口在不增加内存使用量的前提下，能够显著地减少窗口的滑动时间，提高数据流处理系统的效率。

今后的进一步工作是将可重写循环模型集成到现有的数据流处理系统中，加快数据流处理系统的底层处理。

参 考 文 献

1 Anita D, Jasnusz G. Conceptual Modeling of Computations on Data Streams [C]. In: the 2nd Asia-Pacific Conf on Conceptual Modeling(APCCM2005). Newcastle, Australia, 2005. 43～48

2 Babcock B, Bahu S, Dater M, et al. Models and Issues in Data Stream Systems [C]. In: Proceedings of the 21st ACM Symposium on Principles of Database Systems, 2002. 1～16

3 Keogh E. Chakrabarti K, Pazzani M. Locally adaptive dimensionality reduction for indexing large time series databases [C]. In: Proc. of ACM SIGMOD Conference on Management of Data, 2001. 151～162

4 Keogh E, Pazzani M. A simple dimensionality reduction technique for fast similarity search in large time series databases [C]. In: 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Kyoto, Japan, 2000. 122～133

5 Muthukrishnan S. Data streams algorithms and applications [C]. In: Proc. of the 14th Annual ACM-SLAM Symposium on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2003. 413～413

6 Rajeev M, Jennifer W, Arvind A, et al. Query Processing, Approximation, and Resource Management in a Data Stream Management System [C]. In: Proc. of Conf on Innovative Data Syst, Res, 2003. 245～256

7 钱江波，徐宏柄，王永利，等. 多数据流滑动窗口并发连接方法[J]. 计算机研究与发展，2005，42(10)：1771～1778

(上接第 47 页)

Sta→PC 队列组内部的管理策略实质和 PC→Stas 实质是一致的，不再重复。

3.2 自适应和单轮询算法在 CFP 传输帧的比较

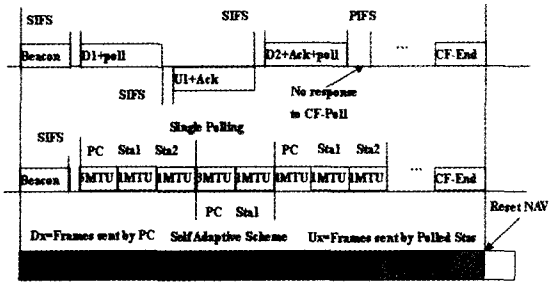

图 3 自适应算法和单轮询算法在 CFP 阶段的帧传输对比

如图 3 所示，PC 经过 PIFS(PCF Inter-Frame Space)间隔后获取信道控制权，发送信标(Beacon)帧，CFP 阶段开始。PC 首先对节点 1 进行轮询，并捎带发送节点 1 的下行数据帧 D1，节点 1 发送上行数据帧 U1，并捎带对 D1 的应答 Ack。PC 继续对节点 2 轮询，捎带发送给节点 2 的下行数据帧 D2 和对 U1 的 Ack。节点 2 没有反应(可能由于数据帧丢失、节点 2 处于 PowerSave 模式或者已经移出本 BBS 等原因)。经过 PIFS 间隔后，PC 收回信道控制权节，继续对下一个节点进行轮询。最后 PC 发送 CF-End，结束 CFP 阶段，各无线节点重置网络分配矢量 NAV(Network Allocation Vector)，进入 CP 阶段。

如果采用自适应调度算法，那么整个帧传输过程是动态的。信标帧之后，进行第一轮服务，PC→Stas 队列先进行数据传输，可以传输 5 个 MTU 数据，之间每个 MTU 传输数据间隔为 SIFS(图中没有画出 SIFS，以下同)。服务 PC 之后，间隔 SIFS，进入 Stas→PC 队列组中的第一个服务队列 Sta1。在传输完 1 个 MTU 之后，调度器将使用权交给 Sta2，传输 1 个 MTU。处理完所有传输的站点之后，Stas→PC 队列组里面有站点离开 BSS，那么在进行的第 2 次服务当中，PC 队列服务完毕之后，处理单独的 Stas→PC 的队列，也即 Sta1 消息队列。在处理完毕 Sta1 队列此次服务，又有新的移动站点加入 BSS，并有数据传输，那么在新一轮服务中，又会对新加入的站点形成的队列及时进行服务。在整个 CFP 过程中，PC 队列在每个服务周期之内，处理的数据是动态变化的。

两个算法在帧传输的过程中，相同点是以信标帧作为开始，PC 发送 CF_End 作为 CFP 的结束。不同点在于，自适应算法可以更加公平地让有数据传输的站点获得传输数据的机会，降低了平均包延迟，并且传输的效率也比单轮询传输更好，而轮询浪费了很多时间用在查询站点是否有数据要传输上。同时，在系统吞吐量上面，由于自适应算法采用了队列管理方式，在实时要求高的情况下和大规模数据传输的情况下，可以缓解网络拥塞，提高性能。

总结 本文在总结已有 PCF 算法的基础上，提出了一个自适应调度算法，定性的分析出了新算法在帧传输方面 QoS 的优越性，对 CFP 阶段多站点高速率传输情况为 PCF 算法提供了一套解决方案。采用队列管理 PC 和站点以及采用合适的缓冲区丢包策略将会是 PCF 算法的一个趋势。

参 考 文 献

1 IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physic Layer (PHY) Specifications [S]. ISO/IEC 8802-11:1999(E), Aug 1999

2 Ganz A, Phonphoem A, GANZ Z. Robust superpoll protocol for IEEE 802.11 wireless LANs [J]. In: Proc. of Military Communications Conference, Oct 1998, 2: 570～574

3 Ziouva E, Antonakopoulos T. Improved IEEE 802.11 PCF Performance Using Silence Detection and Cyclic Shift on Stations Polling [J]. IEE Proc-Commun, Feb 2003, 150(1): 45～51

4 Lo N S C, Lee G, Chen W T. An efficient multipolling mechanism for IEEE 802.11 wireless LANs [J]. IEEE Trans Comput, Jun 2003, 52(6): 764～778

5 Lan YiWen, Chen JyhCheng. Asymptotic weighted fair queuing (AWFQ) for IEEE 802.11 point coordination function (PCF) [J]. In: Consumer Communications and Networking Conference, 2006. 2006 3rd IEEE. Vol 2. Jan 2006. 823～827

6 Yeh JingYuan, Chen Chienhua. Support of multimedia services with the IEEE 802-11 MAC protocol [J]. In: Communications, 2002, ICC 2002. IEEE International Conference on, Vol 1. 28 April-2 May 2002. 600～604

7 Deng D J, Chang R S. A priority scheme for IEEE 802.11 DCF access method [J]. IEICE Trans Commmnun, vol E82-B, Jan 1999

8 Ma X, Du C, Niu Z. Adaptive polling list arrangement scheme for voice transmission with PCF in wireless LANs [J]. In: Commmunications, 2004 and the 5th International Symnposiium on Muilti-Dimensional Mobile Commulnications Proceedings. The 2004 Joint Conference of the 10th Asia-Pacific Conference on, vol 1. Aug 2004

9 Demers A, Keshav S, Shenker S. Analysis and Simulation of a Fair Queueing Algorithm [J]. In: Proc. ACM SigComm 89, Austin, TX, 1989

10 Parekh A K, Gallager R G. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case [J]. IEEE/ACM Transactions on Networking, 1993, 1: 344～357

11 Floyd S, Jacobson V. Random Early Detection Gateways for Congestion Avoidance [J]. IEEE/ACM Transactions on Networking, 1993, 1(4): 397～413

12 Braden B. Recommendations on Queue Management and Congestion Avoidance in the Internet [S]. RFC2309, 1998-04

13 Feng W C, Kandlur D, Saha D, et al. A Self-configuring Red Gateway [C]. In: Proc. of IEEE Infocom, New York, USA, 1999. 1320～1328

14 Jiang Y, Tham C, Ko C. Providing quality of service monitoring: challenges and approaches [J]. International Journal of Network Management, 2000, 10(6): 323～334

15 Trimintzios P. A management and control architecture for providing IP differentiated services in MPLS-based networks [J]. IEEE Communications Magazine, 2001, 39(5): 80～88