# 利用空间优化的增强学习Sarsa改进预取算法

梁 媛 袁景凌 陈旻骋
(武汉理工大学计算机科学与技术学院 武汉 430070)

**摘 要** 数据中心是高性能计算机的集群中心,CPU集群运行繁忙,不规则的数据结构和算法频繁使用,使得大多数基于时空局部性的预取技术不再适用。文中引用语义局部性的概念,使用增强学习Sarsa算法来近似语义位置,预测不规则数据结构和算法未来的内存访问。由于状态空间和动态空间过大,采用Deep Q-learning方法优化状态-动作空间,将新状态与旧状态拟合,相似则采取相似的做法,从而提高泛化能力。在标准数据集SPECCPU 2006上的实验证明,所提方法的泛化能力强,能够有效提高Cache的命中率。

**关键词** 预取技术,语义局部性,Sarsa,Deep Q-learning,状态-动作空间优化
**中图法分类号** TP302 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.03.048

## Prefetching Algorithm of Sarsa Learning Based on Space Optimization

LIANG Yuan YUAN Jing-ling CHEN Min-cheng
(School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China)

**Abstract** As the cluster centers for high-performance computers, data centers are busy with CPU clusters. Irregular data structures and algorithms are frequently used, so that most prefetching technologies based on spatio-temporal locality are no longer applicable. This paper referred to the concept of semantic locality, used the reinforcement learning Sarsa algorithm to approximate semantic locations, and predicted irregular data structures and future memory accesses of algorithms. Due to the large state space and action space, this paper used Deep Q-learning method to optimize the State-action space to fit the new state with the old one and took a similar approach if the two states are similar to improve the generalization ability. The experiment on the standard data set SPECCPU 2006 proves that this method has a wide generalization ability and can improve the Cache hit rate effectively.

**Keywords** Prefetching technology, Semantic locality, Sarsa, Deep Q-learning, State-action space optimization

## 1 引言

内存访问延迟一直是限制处理器的主要问题,尤其是在高性能计算机集群的数据中心中。庞大的CPU集群运行十分繁忙,爆炸式增长的数据量导致了大量不规则的数据结构与算法的应用。随着数据结构的不断复杂化,图、二叉树、邻接矩阵以及表示图形的邻接矩阵、基于数组的二叉树等在空间上往往表现得十分不均匀。因此,基于时空局部性的预取方法也就不再适用。

目前国内外在预取方法上已有大量的研究,但大多是基于空间局部性以及时间局部性(计数器)[1-3]。传统的Next-line[4]预取以及改进的Next-N-line预取技术都是基于空间局部性的预取方法,即每次访问某一个块时就预取下一个顺序块,算法简单,硬件开销小,但无效率很高。Markov模型[5]的预取策略是最基础的时间局部性的预取策略,该策略利用一个表记录给定地址的下一个潜在访问对象来实现Markov预取器,由于相关性流的长度较大,其需要非常大的额外存储空间来存储相关流。Somogyi等[6]提出了以空间关系为目标的空间存储流(SMS)预取器,它使用上下文数据(PC和区域偏移量)来识别不一定连续的空间模式。在此之后,他们将研究扩展至使用时空关系为目标的时空存储流预取器[7]。这3类传统的预取方法在规则数据结构(物理结构与逻辑结构一致)及算法上都有较好的表现。但随着数据结构的复杂化,其物理结构与逻辑结构不一致,空间表现上也不再连续,以上3类传统的预取方法越来越不能满足高性能计算的要求。于是,Peled等[8]提出了用语义局部性来替代传统的时空局部性,基于上下文关系(Context-based),使用上下文老虎机模型(Contextual bandits)近似语义位置,取得了显著的Cache命中率。

到稿日期:2018-01-26 返修日期:2018-04-01 本文受国家自然科学基金(61303029),湖北省自然科学基金重点类项目——创新群体项目(2017CFA012),湖北省技术创新专项重大项目(2017AAA122)资助。

梁 媛(1994—),女,硕士生,CCF学生会员,主要研究方向为机器学习,E-mail:lyuan0416@whut.edu.cn;袁景凌(1975—),女,教授,博士生导师,CCF会员,主要研究方向为绿色计算、机器学习和数据挖掘,E-mail:yuanjingling@126.com(通信作者);陈旻骋(1990—),男,博士生,主要研究方向为机器学习、数据挖掘。

语义局部性[8]是一种由数据结构或遍历算法的语义所反映的数据对象之间的关系的数据局部性的高级抽象概念。例如图 1 是将二叉树放在数组中，3 和 2，4 在空间上是连续的，而 3 却和 1，6，7 在逻辑上是连续的。语义局部性是内存访问的特征，如果访问时是通过给定的操作序列，即程序执行的顺序是从一个数据元素到另一个数据元素，那么访问便是相邻的。它以一种与实际数据布局无关的方式来捕捉数据元素之间的关系，是程序语义所固有的，是空间局部性的范化。

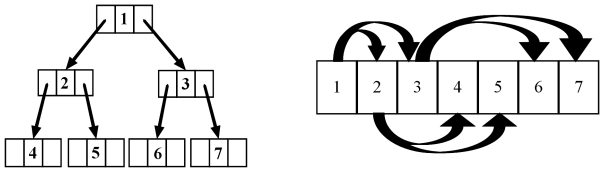


图 1 二叉树的节点位置关系图

Fig. 1 Node location diagram of binary tree

本文根据语义局部性，使用增强学习近似语义位置，预测未来的内存访问，主要工作如下：1）基于上下文关系，利用机器的硬件属性和软件属性作为增强学习 Sarsa 算法的特征值来近似语义局部性；2）采用 Deep Q-learning 方法，优化动作空间和状态空间，并在 gem5 模拟器上对其进行评估。

与现有的研究相比，该方法将新状态与旧状态拟合，相似的状态采取相似的输出动作，泛化能力强，能够有效提高 Cache 的命中率。

## 2 基于 Sarsa 的预取算法

### 2.1 Sarsa 算法

Sarsa 算法[9]属于强化学习算法中的 on-policy 算法，它估计的是动作值函数（$Q$ 函数）而非状态值函数（$V$ 函数）。也就是说，Sarsa 算法估计的是策略 $\pi$ 下，任意状态 $s$ 上所有可执行的动作 $a$ 的动作值函数 $Q_\pi(s_t,a_t)$。Sarsa 的动作值函数的公式如下：

$$Q(s_t,a_t) \tag{1}$$

其中，$Q(s_t,a_t)$ 代表 $t$ 时刻的 $Q$ 值；$s_t$ 为状态；$a_t$ 为当前所采取的动作；$s_{t+1}$，$a_{t+1}$ 则分别是后续的状态和动作；$r_t$ 为奖励函数值；$\alpha\in[0,1]$ 为学习率；$\gamma\in[0,1]$ 称为折合因子，表明了未来的回报相对于当前回报的重要程度。

### 2.2 基于 Sarsa 的预取算法

将 Sarsa 算法应用于 CPU 预取上，CPU 的上下文信息作为算法中的 State 集，而 action 则是预取器下一步准备预取的内存地址。状态集包括指令指针、最近内存访问历史、分支历史、提前加载的数据、类型信息、存储在普通寄存器中的数据[8]等，具体如表 1 所列。

学习机制只能接受在有效预取距离内的语义关系。如果 CPU 马上就要用到某一数据，那么预取器再去抓取这一数据，这一行为此时已经是无意义的；同样地，如果我们将 CPU 很长一段时间以后需要用到的数据预抓取来，这一行为也是无意义的，这是因为在 CPU 实际产生这一需求时，这些数据很有可能已经从缓存中删除了。

表 1 状态集的列表

Table 1 List of state sets

| State | 描述 | 载体 |
|---|---|---|
| 指令指针 | 指令指针是基于指令指针的预取器最常见的上下文属性。指令指针标识了用于相关负载操作的代码中的特定加载站点 | 硬件 |
| 最近内存访问历史 | 以前的地址可能会以重复出现的模式显示未来的地址，但这项功能有过度本地化学习的风险，必须谨慎使用 | 硬件 |
| 分支历史 | 对当前控制流的提示，在某些情况下，可能指示沿发散数据结构的特定路径 | 硬件 |
| 访问数据历史 | 程序访问中所需的基于全局或者局部上下文中的历史数据 | 硬件 |
| 类型信息 | 对象类型的唯一枚举，允许区分复合结构中的数据元素（如图中的边对顶点） | 硬件 |
| 通用寄存器中存储的数据 | 如果遍历决定取决于一些存储在数据结构外的数据（如被搜索的值），则可能提供遍历路径之间的通道 | 编译器 |
| 地址偏移量 | 对象内部的偏移、指针的内部布局或用于标识数据结构中相邻元素的索引 | 编译器 |
| 引用类型 | 指针运算符（“.”“->”或“*”）、数组索引等 | 编译器 |

如图 2 所示，预取距离表示预取地址和下次用到这个地址之间的距离。为了准确识别 State 与 action 在有效预取距离内的语义关系，预取器将重复奖励相似的预取距离。

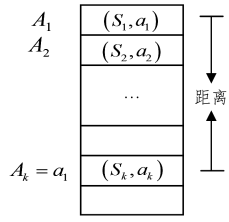


图 2 预取距离

Fig. 2 Prefetch distance

我们在研究中发现，预取距离在 10～90 之间，尤其是在 30 左右时，预取地址的有效性相对较高。太近的预取距离会导致 CPU 需要时，预取器还未进行预取操作；太远的预取距离将导致该内存可能未被采用就被替代算法替换。我们引用 Peled 等[8]提出的如图 3 所示的钟形曲线来表示奖励函数对实际距离进行相应的奖励或惩罚。

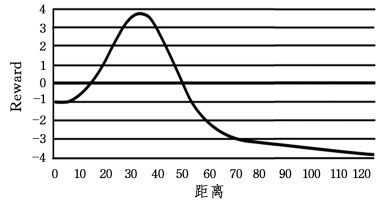


图 3 奖励函数钟形曲线

Fig. 3 Bell-shaped curve of award function

状态空间由所有特征值（即表 1 中所有的行）空间的笛卡尔积组成，动作空间由所有的虚拟地址组成。由于状态空间和动作空间特别大，$Q(s,a)$ 的矩阵异常庞大。

Peled 等[8]使用 List 代替原始的二维数组，来存储 State-action 对和对应的 $Q$ 值。如图 4 所示，List 的最大长度是固定的，同时，List 中只存储已经出现过的 state 和实际采取的 action 对。该方法虽然能解决 $Q$ 值表的存储问题，但无法从相似或相邻的状态中学到信息，且未考虑未出现过的 state 及

其将会采取的 action，泛化能力弱。因此，本文提出了基于 Deep Q-learning 的状态空间优化方法。

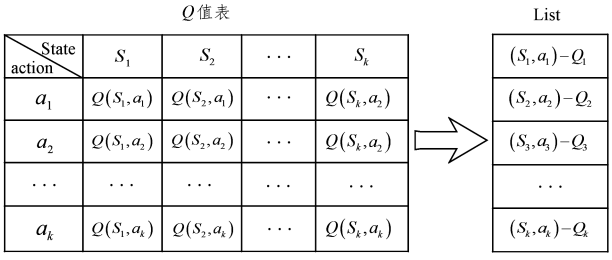


图 4　List 代替二维数组

Fig. 4　Replacing two-dimensional array with List

## 3　Deep Q-learning 状态-动作空间优化

### 3.1　值函数逼近

上述算法存在下列两个重要的缺陷：

1）$Q(s,a)$的矩阵非常大，仅通过反复的测试无法获得足够的样本来遍历每个状态。

2）Peled 等[8]只考虑了已出现过的 state 且忽略了 state 之间的联系，泛化能力弱。

基于以上两点，我们采用近似值函数（value function approximation）[11]的思想对 $Q$ 值表进行近似。如图 5 所示，使用一个函数$\tilde{Q}(s,a;w)$来近似 $Q$ 值表的信息，把 $Q$ 矩阵的更新问题变成一个函数拟合问题，该方法有两个优势：1）对输入任意输入的状态都能输出结果；2）相近的状态可以得到相近的输出动作。

$$\tilde{Q}(s,a;w)\approx Q'(s,a) \tag{2}$$

我们需要更新参数 $w$ 来使得 $Q$ 函数逼近于最优的 $Q$ 值。

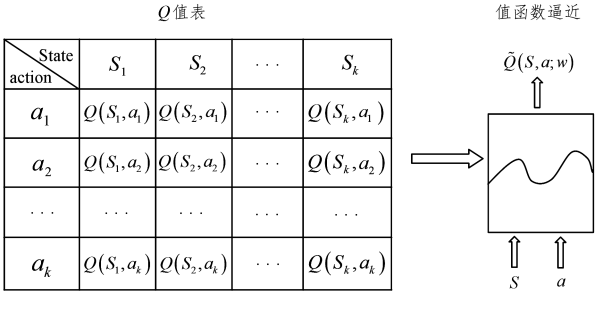


图 5　近似值函数代替二维数组

Fig. 5　Replacing two-dimensional array with approximation function

### 3.2　Deep Q-learning

为了解决上述函数优化的问题，我们使用以下 4 步来完成：

1）采用一个 3 层的神经网络[12-13]作为 $Q$ 值的网络，如图 6 所示，每个网络的神经元为 128 个，每个隐藏层都引入激活函数 ReLU 函数：$f(x)=\begin{cases}0, & x<0\\ x, & x\geqslant 0\end{cases}$，参数为 $w$：$\tilde{Q}(s,a;w)\approx Q^{\pi}(s,a)$。

2）在 $Q$ 值中使用均方误差（mean-square error）来定义损失函数（loss function）：

$$L(w)=E[((r+\gamma \max_{a'} Q(S',a';w))-Q(S,a;w))^2] \tag{3}$$

这里将 $Q$ 的理论值$(r+\gamma \max_{a'} Q(S',a';w))$作为目标。以当前值和理论目标值的均方误差作为该网络的损失函数。

3）计算参数 $w$ 关于 loss function 的梯度：

$$\frac{\partial L(w)}{\partial w}=E[((r+\gamma \max_{a'} Q(S',a';w))-Q(S,a;w))\frac{\partial Q(S,a;w)}{\partial w}] \tag{4}$$

4）使用 SGD 实现 End-to-end 的优化目标，得到最优的 $Q$ 值。

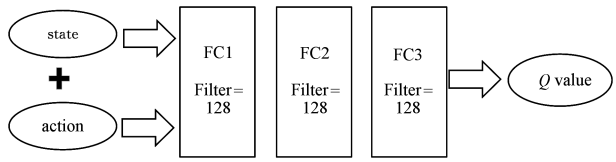


图 6　3 层神经网络模型

Fig. 6　Three-layer neural network model

### 3.3　Multi-Q value table

由于 action 空间过大，如果直接使用 Deep Q-learning 方法计算最优的 $Q$ 值，对于指定的 $s_i$，寻找最大 $Q$ 值对应的 action：$\arg\max_a Q(s_i,a)$的时间复杂度过高。因此，采用 multi-Q value table 方法[14]，找出最大 $Q$ 值对应 action 的大致范围，当 $Q$ 值大于某个较低阈值时，采用 Deep Q-learning 在这一范围中找到最大 $Q$ 值对应的具体 action。

如图 7 所示，保留多个稀疏的 $Q$ 值表，多个 $Q$ 值表在状态空间的各个维度中较第一个 $Q$ 值表 $Q_1$ 相差不同的偏移量。这里以两个属性所组成的状态空间为例，假设 $S_1$ 空间大小为 $m$，$S_2$ 空间大小为 $n$，故状态空间 $S=\{(s_1,s_2)\mid s_1$ 属于符号 $S_1$，$s_2$ 属于符号 $S_2\}$，即需要 $m*n$ 的空间。我们采用 $k$ 个状态空间对这个状态空间进行简化，这两个状态空间的维度颗粒度分别降低为原来的 $1/k$，并各自取 $0,1,2,\cdots,k-1$ 作为其偏移量，则现在只需要 $\frac{m}{k}*\frac{n}{k}*k$ 的空间，对高纬度状态而言其优化程度更高，对于某一具体状态 $S$ 的 $q$ 值，将其取值为各 $Q$ 值表中该状态对应 $Q$ 值的代数平均值：$\frac{Q_1(S)+Q_2(S)+\cdots+Q_k(S)}{k}$。

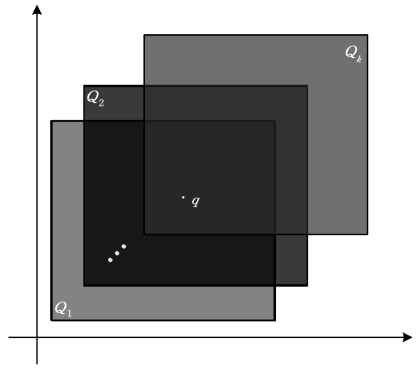


图 7　偏移量为 $k$ 的多个 $Q$ 值表

Fig. 7　Multiple $Q$ value tables when offset is $k$

因此，在训练好 Deep Q-learning 网络和采用稀疏表进行优化得到普通 $Q$ 值表后，我们采用以下两个步骤来寻找最大

$Q$ 值对应的 action：$\arg\max_a Q(s_t, a)$。

1）使用 Multi-Q value table 方法，找出某状态下最大 $Q$ 值对应 action 的大致范围；

2）使用 Deep Q-learning 方法，得到该状态下最大 $Q$ 值对应的 action。

## 4 实验设计与结果分析

本文实验在 Ubuntu16.04 环境下使用配置为 Intel Core i7 7700，RAM 4GB 的计算机运行，使用了 gem5 模拟器＋DRAM sim2 平台作为实验环境，以及 SPECCPU2006 作为实验数据集。表 2 详细介绍了模拟器参数。

表 2 模拟器参数

Table 2 Simulator parameters

| | |
|---|---|
| CST | 1024＊3＊4＝12K |
| $Q$ 值 List | $n$＊2K($n$＝10) |
| 网络参数 | 3＊128＊128＝48K |

我们评估预取器算法是通过研究预取器的准确性以及它对缓存和系统性能的影响。为了对比验证上述 2 个性能指标，将本文提出的预取器与基于上下文(Context)的预取器和空间记忆流算法(SMS)进行了两组对比实验。将所有预取器的尺寸大小都调整为与本文预取器大小相同。

### 4.1 准确性

实验数据集采用由 SPEC 组织推出的 CPU 子系统评估软件 SPECCPU2006[15]。SPECCPU2006 包括了 CINT2006 和 CFP2006 两个子项目，前者用于测量和对比整数性能，后者则用于测量和对比浮点性能。表 3 是基准测试集 SPECCPU2006 的子数据集。

表 3 基准测试集 SPECCPU2006 的子数据集

Table 3 Subset of SPECCPU2006 benchmark set

| 子数据集 | 简介 |
|---|---|
| astar(寻路算法) | 实现了 2D 寻路算法 $A^*$ 的 3 种不同版本 |
| lbm(流体动力学) | 使用 LBM(格子玻尔兹曼方法)模拟非压缩流体 |
| mcf(组合优化) | MCF 是一个用于大型公共交通中的单站车辆调度的程序 |
| omnetpp(离散事件仿真) | 包括约 8000 台计算机和 900 个交换机/集线器，以及混合了各种从 10Mb 到 1000Mb 速率的大型 CSMA/CD 协议以太网络模拟 |
| sphinx3(语音识别) | 语音识别 |

本文评估预取器准确性的方法是以每次预取器访问内存地址的结果好坏为评判标准，结果分为以下 6 种情况。

1）命中预取线：说明预取是有效的。

2）较短等待时间：预取没有命中缓存，但由于持续的预取缩短了等待时间。

3）不及时的预取：预取器预测了内存地址，但没有及时地发送到内存去。

4）未预取：预取器没有预测内存地址。

5）沿用旧地址：这种情况下预取器不需要预测地址。

6）预取未命中：预取器预测了一个错误的地址。

图 8 展示了使用 3 种预取器算法在各个算例下各情况所占的比例。该数据表明，本算法不仅继承了 Peled 等[8]提出的 Context-based 算法在链式离散数据结构上的优势，而且在绝大部分算例上比原本的算法更优。预取器在预测准确率有微弱提高的情况下，较大地缩短了预测地址的等待时间。可以发现，在 SPEC 基准测试集(lbm，omnetpp 和 sphinx)中，这个结论更加明显。本文在预取算法中使用 Deep-Q learning 来预测 $Q$ 值最大的内存地址，收集了程序和系统的上下文信息作为增强学习的状态空间。与传统的根据时空局部性的 SMS 预取算法[7]相比，本文在处理不规则数据结构时更关注数据对象之间的语义局部性，因此预取效果更好。Context-based 方法使用有限长度的 List 只存储已经出现过的 State 和实际采取的 action 对，未考虑未出现的 State 以及相近 State 之间的联系，泛化能力弱；而本文使用 Deep-Q learning 来预测 $Q$ 值最大的内存地址，即使有未出现的 State，也能结合已有的相似 State 预测出有效的内存地址，因此预取效果更好。然而，在处理二维数组这种规则的数据结构时，本文提出的预取算法反而没有传统的 SMS 算法效果好，在 astar 寻址算法这个数据集上，命中预取线的比例较低。

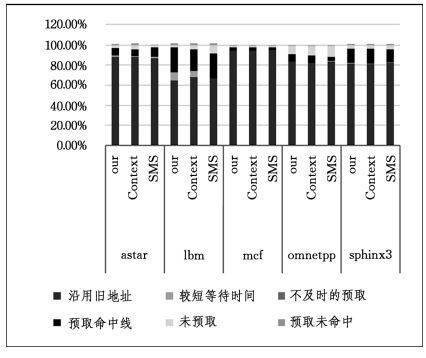


图 8 各预取器算法的准确性结果

Fig. 8 Accuracy results of prefetcher accesses memory address

### 4.2 缓存性能分析

本文通过计算 L1 和 L2 缓存中每 1000 条指令出现的缺失数目（$MPKI$）来评估缓存性能，$MPKI$ 值越低，算法性能越好。$MPKI$ 反映了在回报函数范围内预取算法对于短期和中期延迟访问的有效性。

图 9 和图 10 展示了 3 种预取算法在 L1 和 L2 缓存中的 $MPKI$ 值。在 L1 缓存中，本文提出的预取算法在减轻了 Context-based 算法在部分算例(例如 astar，lbm，mfc)上的性能缺陷的前提下，尽可能地接近传统的 SMS 算法，这些算例多采用连续的数据结构；而剩下的算例(例如 omnetpp，sphinx3)表现出了最佳的性能，明显优于其他两种算法。

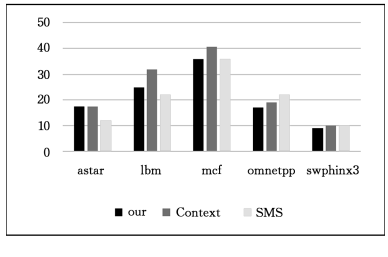


图 9 L1 缓存中 3 种预取算法的 $MPKI$ 值

Fig. 9 $MPKI$ values for three prefetching algorithms in L1 cache

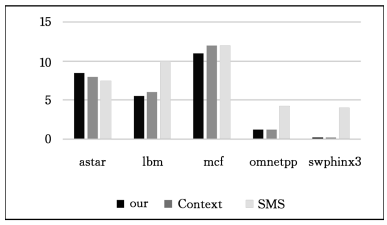


图 10　L2 缓存中 3 种预取算法的 *MPKI* 值

Fig. 10　*MPKI* values for three prefetching algorithms in L2 cache

在 L2 缓存中,本文提出的预取算法与其他两种预取算法相比,*MPKI* 值都较低。平均而言,本文预取算法的 L2 缓存的 *MPKI* 值是 SMS 算法的 2 倍,且略好于基于上下文的预取算法。

综上所述,对于离散链式数据结构,本文采用的预取算法在可接受的时间约束下优于 Context-based 算法,远优于传统的预取算法;对于连续的数据结构,本文采用的预取算法在一定程度上弥补了 Context-based 算法的缺陷,更接近传统预取算法的性能。由于大型数据中心绝大部分采用离散的链式数据结构,利用本文算法能取得最优的效果。

**结束语**　内存访问延迟一直是限制处理器的主要问题,尤其在高性能计算机集群的数据中心中。本文使用增强学习 Sarsa 算法近似语义位置,并结合 Deep Q-learning 方法优化状态空间。实验结果表明,该方法的泛化能力较强,能够有效改善 Cache 命中率。我们后续将研究连续数据与不连续数据的命中率。

## 参 考 文 献

[1] SMITH A J. Cache Memories[J]. Acm Computing Surveys, 1982,14(3):473-530.

[2] WENISCH T F. Temporal memory streaming[D]. Pittsburgh: Carnegie Mellon University,2007.

[3] SOMOGYI S,WENISCH T F,AILAMAKI A,et al. Spatio-temporal memory streaming[J]. Acm Sigarch Computer Architecture News,2009,37(3):69-80.

[4] COOKSEY R N,JOURDAN S J. Method and apparatus for next-line prefetching from a predicted memory address: US, US 7093077 B2[P]. 2006.

[5] JOSEPH D,GRUNWALD D. Prefetching using Markov predictors[C]// Proceedings of the 24th Annual International Symposium on Computer Architecture. IEEE,1997:252-263.

[6] SOMOGYI S,WENISCH T F,AILAMAKI A,et al. Spatial Memory Streaming[C]// International Symposium on Computer Architecture. IEEE,2006:252-263.

[7] SOMOGYI S,WENISCH T F,AILAMAKI A,et al. Spatio-temporal memory streaming[J]. AcmSigarch Computer Architecture News,2009,37(3):69-80.

[8] PELED L,MANNOR S,WEISER U,et al. Semantic locality and context-based prefetching using reinforcement learning[C]// International Symposium on Computer Architecture. ACM,2015: 285-297.

[9] MOZER S,M C,HASSELMO M. Reinforcement Learning: An Introduction[J]. IEEE Transactions on Neural Networks,2005, 16(1):285-286.

[10] IPEK E,MUTLU O,CARUANA R. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach[C]// International Symposium on Computer Architecture. IEEE,2008: 39-50.

[11] SUTTON R S,MCALLESTER D,SINGH S,et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation[J]. Advances in Neural Information Processing Systems,2000,12:1057-1063.

[12] TAN F,YAN P,GUAN X. Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning[C]// International Conference on Neural Information Processing. Springer, Cham, 2017: 475-483.

[13] LIN L M,WANG H,WANG Y X. Sarsa Reinforcement Learning Algorithm Based on Neural Networks[J]. Computer Technology & Development,2006(1):30-32. (in Chinese)
林联明,王浩,王一雄. 基于神经网络的 Sarsa 强化学习算法[J]. 计算机技术与发展,2006(1):30-32.

[14] DURYEA E,GANGER M,HU W. Exploring Deep Reinforcement Learning with Multi Q-Learning[J]. Intelligent Control & Automation,2016,7(4):129-144.

[15] HENNING J L. SPEC CPU2006 benchmark descriptions[J]. Acm Sigarch Computer Architecture News,2006,34(4):1-17.