负载感知的存储子系统调优研究*)

李益民 邢春晓 严 琪 胡庆成 张小虎
(清华大学计算机科学与技术系 北京 100084)

摘 要 存储IO的性能远远低于CPU、内存的性能,而且它们之间的差距还在扩大。因此,对于越来越多的数据密集型应用系统来说,存储系统往往是系统瓶颈。存储的性能不仅与存储子系统体系结构以及子系统中各部件的性能相关,还与系统的工作负载和应用环境相关。负载感知的性能调优指存储子系统通过对负载特征的分析实现对应用环境的动态感知,并根据负载特征动态调整系统运行策略。负载感知的性能调优使得存储子系统能够更合理地调度存储系统资源,从而提高IO性能。

关键词 存储子系统,负载感知,性能调优

Study on Workload-aware Tuning Methods of Storage Subsystem

LI Yi-Min XING Chun-Xiao YAN Qi HU Qing-Cheng ZHANG Xiao-Hu
(Dept. of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract The I/O performance of storage sub-system is far lower than that of CPU or memory, and the gap between them is growing greater and greater. Storage subsystem is still the performance bottleneck of computer systems for data-intensive applications. The I/O performance of storage subsystem is associated with not only the architecture and the components of storage sub-system but workload characteristics and application circumstances as well. The core idea of workload sensitive performance-tuning is that storage subsystem continually analyzes workload characteristics so as to dynamically make sense of the application circumstances and automatically adjusts the systematical running strategy. Storage subsystem can reasonably schedule storage resources and considerably improve IO performance by Workload-Aware performance tuning.

Keywords Storage subsystem, Workload-aware, Performance tuning

1 引言

计算机科学技术的迅速发展和广泛应用推动了社会生产和生活各个领域的快速发展。与此同时,社会生产和生活各个领域导致计算机需要面对的数据量呈指数式增长,极大地推动了数据存储尤其是海量数据存储技术的发展。以服务器为中心的数据存储模式逐渐转化为以数据为中心的数据存储模式。网络附加存储(NAS)和存储区域网络(SAN)成为存储领域近年来的主流技术,它们突破了直接连接存储(DAS)存储结构的限制,不仅提高了存储系统的扩展性、可用性和灵活性,更重要的是带来了数据IO性能的大幅度提高。

然而相比较而言,目前外部存储设备的IO性能相对于CPU、内存和网络来说要低6个数量级以上[1],并且由于受机械部件性能的限制,存储设备的访问速度每年只能提高7%~10%,而现代处理器和内存的处理速度正以每年50%~100%的速度提升,存储设备IO性能的提高速度远远低于CPU和内存。对于越来越多的数据密集型应用来说,存储设备IO性能依然是计算机系统整体性能的瓶颈。要提高计算机系统的整体性能,改善相关应用的服务质量,对存储子系统进行调优是条非常有效的途径。

传统的存储子系统试图通过一些固定的、静态的策略来优化IO性能,而对自身所处的应用环境和负载特征并不关心。实际上,在不同的应用环境下,施加到存储子系统的IO负载往往呈现出不同的模式和特征[2,3];而且,在同一应用环境中,随着时间的推移和用户群的变化,IO负载模式和特征也会不断改变。存储子系统中很多部件的性能对于负载特征是非常敏感的,因而感知负载的变化并有针对性地对系统行为进行调节,能使存储子系统各部件的协作更加协调,存储资源的分配更加合理,有助于存储子系统IO性能的提高。

负载感知的自调优存储子系统是一个能动态感知环境变化、自动提取负载特征,并根据分析自动实施策略调整的自调优系统。本文对自调优存储子系统中的负载敏感部件进行分析,并提出负载感知的调优方法。对其相关技术进行详细和深入介绍和对比研究。本文以下共有三个部分:第二部分进行存储子系统的典型结构及其负载敏感部件的性能分析;第三部分介绍负载感知的存储系统性能调优方法;最后是文章的总结。

2 存储子系统典型结构及其IO性能分析

2.1 存储子系统典型结构

存储子系统的主要任务就是将后端的物理存储设备向外部世界表现成独立的、可管理的逻辑存储设备。一个典型的

*)国家CNGI产业化类项目(分项CNGI-04-5-1D)、中国博士后科学基金(20060390069)资助,国家科技撑计划(2006BAH02A00)资助。**李益民** 硕士研究生,主要研究方向是存储性能优化;**邢春晓** 博士、教授,主要研究方向为数据库技术、软件工程和海量信息管理;**严 琪** 硕士研究生,主要研究方向为网络存储、射频识别;**胡庆成** 硕士研究生,主要研究方向是数据存储;**张小虎** 硕士研究生,主要研究方向是数据挖掘、软件工程等。

存储子系统包含前端网络、存储控制器、后端网络和物理存储设备四个部分，其结构如图 1 所示。

前端的存储网络连接主机和存储子系统。前端网络可能是光纤通道(FC)、IP 网络或是 InfiniBand。前端网络接口的数量和速度决定了存储子系统的数据 IO 的速度。多接口能实现负载均衡，也能作为冗余链路来实现高可用性。

存储控制器是存储子系统中的核心部件，负责协议的解析、逻辑存储地址到物理存储地址的映射、数据的搬运、数据的保护等功能。为了实现高性能，控制器在实现上一般使用高性能的 CPU、高速的大容量的内存以及针对快速处理数据做了优化的专用集成电路来协同工作。现代的存储控制器已经具备相当强的处理能力。

后端存储网络用于连接存储控制器和后端的存储设备(如磁盘、磁带驱动器)，它可能是一个 FC 环或是一个并行的小型计算机接口(SCSI)总线或是 ATA、SATA 等总线。

真正存储数据的设备是后端存储单元，它们一般是带有 SCSI、FC、ATA 或者 SATA 接口的磁盘驱动器或磁带驱动器。硬盘的转速、平均寻道时间以及接口速度以及磁带机持续传输速率都会影响存储子系统的 IO 性能。

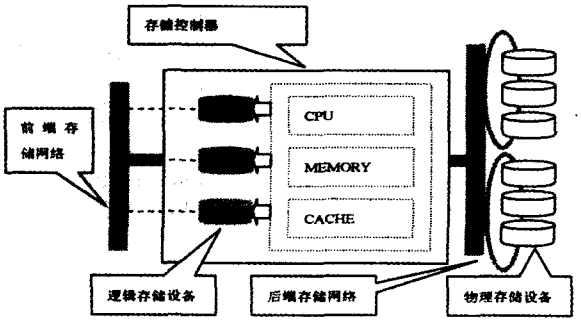

图 1　典型的存储子系统体系结构图

2.2　影响存储子系统 IO 性能的关键因素

2.2.1　数据布局策略

存储子系统协同多个后端存储设备工作，而对于单个存储设备如磁盘驱动器来说，IO 操作是串行的，因此要充分利用所有这些存储设备的性能和容量，就必须将数据分布到尽可能多的存储设备上去，这样就可以对数据进行并行访问。数据的条带化(Striping)就是指对数据分块并将数据分布到不同的磁盘驱动器上。条带化通过引入并行访问使得数据顺序访问的吞吐率大大提高。但是，磁盘的数量越多，磁盘损坏和数据丢失的风险也就越大。因此条带化的数据必须使用镜像或者校验方式来进行保护。冗余磁盘阵列(RAID)[4]就是一个综合了条带化、镜像以及校验的数据分布技术。按照数据分布方式的不同，RAID 分成多个级别。各个 RAID 级别在数据冗余程度、性能等方面各有不同。例如 RAID1 读写性能和可靠性都比较好，但由于其采用全冗余方式，磁盘利用率低、成本高。而 RAID5 读性能好、磁盘利用率高，但由于存在读-更新-写问题，因此不适合写密集型的负载。

很多 RAID 级别都使用了条带(Stripe)来组织数据，将逻辑上连续的数据分布到多个磁盘。条带单元(Striping Unit)指单个磁盘设备上的连续的数据块，条带单元决定了一个逻辑的访问请求如何被分割成磁盘请求。条带单元的大小(Striping Unit Size 以下简称 SUS)是影响 RAID 性能的一个关键因素。Miller 的研究表明，在科学计算中，如果 SUS 的值是最优值的一半或 2 倍时，通常会有 20% 的性能损失[5]。SUS 的值的最优值与负载特征密切相关。简单来说，SUS 设置过大，会导致负载请求无法有效地分布到多个磁盘；而 SUS 设置过小，则访问相同数据量需要的访问次数就会增多，导致额外的开销。

RAID 级别和 SUS 是数据布局中的两个关键因素，它们的最优配置是与负载密切相关的。换句话说，我们需要根据负载来调整数据布局，从而对存储性能进行优化。

2.2.2　数据缓存方法

如图 1 所示，存储子系统常使用缓存(cache)来构成二级或多级的存储层次。由于缓存的访问速度与存储设备相比要快得多，因此可以用缓存来避免缓慢的磁盘访问。对于读取操作来说，如果需要读取的数据已经存在于缓存中，那么无需存储设备的机械操作就能读取数据。对于写入来说，数据可以高速地写入缓存，等系统空闲的时候再真正写入存储设备。最理想情况下，缓存和磁盘组成的多级存储的容量是磁盘的容量，访问速度是缓存的速度。然而在实际情况中，由于需要的数据并不一定在缓存中，因此缓存的命中率是影响多级存储系统的性能的关键，而缓存的调度策略和缓存的容量直接影响缓存的命中率。

传统的存储系统根据访问的局部性原理，使用 LRU 等经典缓存调度算法来进行缓存的调度。这种传统的方式存在一些固有的问题，制约了调度的效率的提高。首先，访问的局部性原理是一个统计规律，LRU 等算法也是一种固定的、非自适应的算法，在实际的某些系统(或是某些场景)中存储 IO 请求不遵循该规律时，缓存的调度就可能出现低效的情况。其次，从主机到存储子系统，甚至在存储子系统内部都存在多个级别的缓存，但多个缓存之间也无法交换信息，无法协同工作。因此这些不同层次的缓存中保存的内容会出现重复，有效的缓存容量将小于总的缓存容量，造成缓存资源的浪费。最后，数据预取根据数据访问的顺序性进行数据预读取取，数据预取需要付出时间上和空间上的开销。一旦数据的访问不满足顺序访问的条件，预取就会取到并不需要的数据，既付出了时间开销，又浪费了缓存空间。

3　负载感知的存储系统性能调优

3.1　存储负载定义及描述

本文所指的负载指存储 IO 负载，是主机向存储设备发出的 IO 请求的集合。假设时间间隔为 t 的范围内，存储设备收到 k 个存储 IO 请求。请求可用如下的向量来描述：

$$\overline{x_i}=(x_{i,1},x_{i,2},x_{i,3},x_{i,4}),0\leqslant i<k$$

其中 $x_{i,1}$ 到 $x_{i,4}$ 分别表示 IO 请求的类型、请求地址、请求长度、到达时间。请求类型取值 0 或 1，分别代表读请求和写请求。那么读写负载所占比重分别为

$$Rreq\_rate=\frac{\sum_{i=0}^{k}(1-x_{i,1})}{k}$$

$$Rreq\_rate=\frac{\sum_{i=0}^{k}(1-x_{i,1})}{k}$$

平均请求长度为

$$average\_req\_size=\frac{\sum_{i=0}^{k}(x_{i,3})}{k}$$

负载的并发度(concurrency)是负载的一个重要特性，它

是指某个时刻，计算机系统中并发地进行磁盘请求的进程的平均数量。并发度的计算可转化为时间间隔内到达请求的平均数。对于请求集合中的每两个向量 $\vec{x_i}$ 和 $\vec{x_j}$，定义距离

$D_{i,j}=|x_{i,3}-x_{j,3}|$

对集合中向量进行聚类，得到

$\{C_1 C_2 \cdots C_m\}$

设聚类中的元素个数为 $Num(C_i)$，则并发度

$$concurrency=\frac{\sum_{i=1}^{m} Num(C_i)}{m} \quad (1)$$

3.2 负载感知的存储子系统体系结构

负载感知的存储子系统中引入了一个负载感知部件，负载感知部件可以感知主机系统的各个层次的负载特征，并根据负载特征调整优化策略。通过将系统资源进行调整适应负载，来实现对系统性的优化。负载感知的存储子系统的体系结构如图 2 所示。

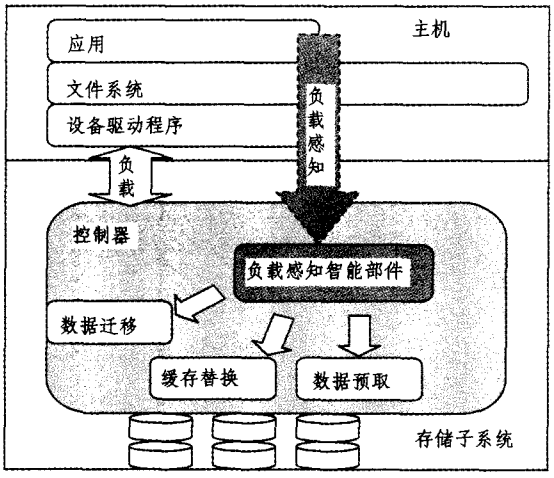

图 2 负载感知的存储子系统示意图

3.3 负载感知方法

3.3.1 挖掘方法

通常情况下，数据以块的形式存储在物理存储设备上，由文件系统管理和组织，应用程序通过文件系统来访问数据。应用程序使文件与文件之间存在某些联系，而文件使构成数据块之间存在联系，对于应用程序直接使用原始分区的情况，则是应用程序直接使某些数据块之间发生联系。因此，在存储设备中，许多数据块之间并非彼此独立，而是存在相关联系[6]的。相互关联的这些数据块很有可能在一个访问流中被先后访问，如果知道这种相关性，我们就可以在读取当前数据块时将与当前数据块相关性大的数据块预取到缓存，或是保留在缓存中不被调出，相对于传统的缓存调度算法，这种方法能够更准确地进行预读取和缓存替换，缓存调度效率更高。

然而现有的存储接口是简单的块访问接口，存储应用只能进行基于块的读写操作，无法向存储设备提供访问模式及数据语义的信息。而存储设备也只能在块级别上管理数据，无法直接知道这些块之间可能存在的语义联系。在不改动存储访问接口的情况下，可以使用一些数据挖掘方法[6]，挖掘出数据 IO 负载的访问模式以及数据块之间的联系。

3.3.2 OSD 方法

自从 1956 年第一块磁盘驱动器诞生以来，存储基础设施的容量和复杂性都有了巨大的提高，传统的块操作接口已经成为存储设备功能增强和性能提升的制约因素。为了对简单的存储块接口进行改进，能向下层存储传递更加丰富的信息，更充分地利用磁盘驱动器中的处理能力，学术研究人员和磁盘制造商都已经在基于对象的存储(OSD)[7,8]上做了一些有价值的工作。基于对象的存储设备使得主机与设备之间的接口不再是简单的块接口，而是对象。对象是一个存储设备的逻辑抽象，包含访问数据的方法、数据描述特征以及安全策略等。通过丰富的对象接口，存储设备能获取到真正使用该存储系统的用户和存储应用的行为、特征和模式等信息。

目前存储工业协会(SINA)已经做了大量 OSD 模型和接口标准化的工作，将 OSD 专用命令扩充到 SCSI 命令集的草案[9]已经被批准，但真正使用 OSD 的产品尚未出现。

这两种负载感知的方法各有其优缺点。挖掘方法无需在接口上进行变动，对主机是透明的。但是，对 IO 请求数据的挖掘需要较大的时间和空间上的开销。OSD 方法在接口上支持负载附加信息的传递，但标准的制定和得到各方面的支持还需要较长的时间。

3.4 基于负载感知的调优方法

3.4.1 负载感知的数据迁移

RAID 的配置会给存储管理带来一定的负担，而且不合理的配置会造成性能问题。同时，系统负载会因随着时间的推移和用户群行为的变化而发生变化，固定的配置无法适应负载的改变。为了提高存储设备的动态适应性，需要根据负载特征进行数据迁移，以实现数据布局对负载的重新适应。负载感知的数据迁移，就是根据负载特征，对数据分布进行重新调整。

由于数据的迁移需要较大开销，因此负载感知的数据迁移需要确定迁移目标、迁移方法和迁移时机的问题。

迁移目标指数据迁移到何种数据分布。由于 RAID1 具有明显的高性能和高成本的特性，而 RAID5 虽然磁盘利用率较高，但在小写负载下性能不佳，许多研究都选择在这两个级别之间进行数据迁移，希望获得这两种数据布局形式的优点。如 Wilkes 等提出了 HP AutoRAID[10]控制器的设计，它能监视负载的情况并根据工作负载来自动实现数据在 RAID5 和 RAID1 之间迁移。这样就能综合 RAID5 的磁盘利用率高以及 RAID1 性能高的优点。

除了 RAID 级别，SUS 也是数据迁移中的一个重要参数。最优的 SUS 值可以根据负载特征进行计算。Chen 和 Patterson[11]使用并发度来描述系统负载，他们认为 SUS 值的选择主要依赖工作负载的并发度，只要并发度已知，就能求出一个在这种负载下一个最优的 SUS 值。对于 RAID1，他们提出了一个公式，用于近似地计算最优的 SUS 值：

$$stripeunit=S\times AT\times ATR\times(concurrency-1)+0.5kB \quad (2)$$

公式中 S 为常数，AT 为平均的磁头寻道并移动到数据所在位置的时间，ATR 指平均的数据传输速率。AT 与 ATR 的乘积也常被称为磁盘性能积(Disk Performance Product)，concurrency 指并发度，可由公式(1)计算得出。0.5kB 是一个磁盘扇区的容量。

Simitci 和 Reed[12]使用单位时间内的到达率来描述负载，并提出了一个基于负载的条带化模型。他们推导了一个公式来计算最优的 SUS 值：

$$stripeunit=\frac{Average\ Re\ questSize}{RequestWidth\times NumOfDisk}$$

对于 RAID5 的条带单元大小的选择，Chen 和 Lee 的研

究[13]表明，对于读负载，最优的 SUS 值随整列中磁盘数的增加而减小，而对于写负载，最优的 SUS 值随阵列中的磁盘数的增加而增加。他们建议在没有详细的负载描述信息的情况下，RAID5 磁盘阵列的 SUS 设为磁盘性能积的 1/2，即

$$\frac{AT \times ATR}{2}$$

数据的迁移方法通常是用一个分配位图维护逻辑存储到物理存储分配的映射关系，数据在物理存储之间迁移，同时修改映射关系，对上层透明。逻辑存储与物理存储的映射关系如图 3 所示。

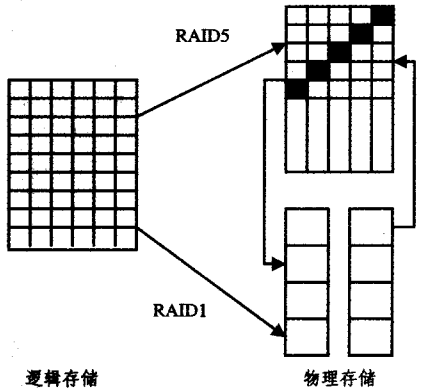

图 3　逻辑存储到物理存储的映射

数据的迁移时机对于系统性能的影响也很大。一方面由于迁移开销较大，数据的迁移不应该频繁发生；另一方面如果在系统繁忙时迁移将会影响存储子系统的性能，因而迁移应该在系统空闲或者轻载时进行。

3.4.2　负载感知的缓存替换

负载感知的缓存调度试图从负载信息中动态挖掘出数据块之间可能存在的相关性、寻找频繁出现的、相对固定的访问模式，从而能预测将要访问的数据块，使得缓存替换和数据预取更加智能。负载感知的缓存首先采用频繁序列(Frequent Sequence)的挖掘算法[14]挖掘频繁出现的请求序列，根据序列计算数据块之间的相关性。

在现有的存储系统体系结构下，文件系统被用于管理文件数据。存储应用通过文件系统访问文件数据。因此存储设备中的数据往往能通过文件系统和存储应用发生某种联系，任意两个数据块 A 和 B 之间可能发生的联系有以下三种：

1)属于同一个文件的不同部分。

2)不属于同一个文件，但属于某个存储应用需要访问的不同的数据文件。

3)数据块之间相关性很小。

对于 1、2 两种情况，数据块之间很有可能存在某种固定的先后访问的顺序，即数据块 A 被访问的事件发生后，数据块 B 被访问的事件很可能发生。因此，在数据块 A 被访问后，如果 B 在缓存中，则 B 不应该被调出，如果 B 不在缓存中，B 应该被优先预取到缓存。负载感知的目的，就是要找到可能存在的数据块访问的序列关系。

例如，对于在 t 时间间隔内收到的 k 个存储 IO 请求，将其访问地址按其到达时间进行排列，如下所示：

{a, b, c ,a, g, d, b, c, g ,a d…}

可以使用频繁序列挖掘算法来挖掘出频繁出现的访问模式。频繁序列是指集合中出现次数超过一定阈值的子序列。子序列出现的次数称为支持度。在上面的例子中进行频繁子序列的挖掘，可以得到{abc}{gd}两个支持度为 2 的子序列。

根据频繁子序列可以计算数据块之间的相关性，与当前访问数据块有强相关性的数据块将不允许被替换出缓存。这点可以用一个使用遮罩的改进 LRU 算法来实现，设频繁序列保存在 FS 中，相关算法的伪码描述如下：

算法 1：

```
LRU_with_mask()
Input: request stream
Output
    For every x in request stream
Case I: x is in CACHE
Cache hit has occurred
Case II: x is not in L
      If L is not full
        load (x).
      else
          find the LRU page r
          if r is in mask
            replace r with x
            else
              find the LRU page in page r'
              replace r' with x
              endif
            endif
```

3.4.3　负载感知的调优方法小结

负载感知的数据迁移根据负载访问特征来调整物理存储设备上的数据布局策略，使得数据的分布主动地适应负载。因此，这种调整是一个缓慢的渐进的过程。一旦负载特征发生变化，重新调整的动作也将是比较迟钝的。负载感知的缓存调度通过挖掘访问序列的模式来对将来访问的数据进行预测，使得缓存替换过程中缓存能避免将将要访问的数据替换出缓存。因此使得缓存替换更加准确，并能提高缓存的命中率。两种调优方法分别针对不同的层次，既可单独使用，也可组合使用。

结束语　存储子系统的 IO 性能依然是计算机系统整体性能的瓶颈，要想提高计算机系统的整体性必须对存储子系统的 IO 性能进行优化。存储的性能不仅与存储子系统体系结构以及子系统中各部件的性能相关，还与应用环境和 IO 负载特征相关。存储子系统中存在一些负载敏感的部件，负载的模式和特征会对存储子系统的性能造成较大影响。而传统的存储子系统无法感知负载，只能使用固定的策略来适用所有的环境，无法动态针对负载进行系统资源的调度策略进行调整和性能上的优化。文章对存储子系统中负载敏感部件性能的影响进行了分析，并介绍了根据负载特征对数据分布策略和缓存调度策略优化的方法。

参 考 文 献

1　Griffin J L, Schlosser S W, Ganger G R, et al. Operating System Management of MEMS-Based Storage Devices. In: Proceedings of the 4th Symposium on Operating Systems Design and Implementation 2000. 227～242

2　Arlitt M F, Williamson C L. Web server workload characterization: the search for invariants. ACM SIGMETRICS Performance Evaluation Review, 1996, 24(1): 126～137

3　Smirni E, Reed D A. Workload Characterization of Input/Output Intensive Parallel Applications. In: Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, 1997. 169～180

4　David A P, Garth G, Katz R H. A Case for Redundant Arrays of Inexpensive Disk (RAID). In: International Conference on Management of Data (SIGMOD) , 1988. 109～116

5　Miller E L, Katz R H. Input/Output Behavior of Supercomputing Applications. In: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, 1991. 567～576

6　Li Z, Chen Z, Zhou SaY C-Miner. Mining Block Correlations in Storage Systems. In: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, 2004. 173～186

7 Mesnier M, Ganger G R, Riedel E. Object-based storage. Communications Magazine, IEEE 2003,34(8):84～90
8 Qin L, Feng D. Active Storage Framework for Object-based Storage Device. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06) 2006, 2:97～101
9 Technology I. SCSI Object-Based Storage Device Commands (OSD), working draft Project T10/1355-D,2004
10 John W, Richard G, Carl S, et al. The HP AutoRAID hierarchical storage system. ACM Trans Computer System, 1996, 14(1): 108～136
11 Peter M C, David A P. Maximizing performance in a striped disk array. In: Proceedings of the 17th annual international symposium on Computer Architecture. Seattle, Washington, United States: ACM Press,1990
12 Reed D. Striping in aRAID Level 5 Disk Array Performance Evaluation Review. A quarterly publication of the Special Interest Committee on Measurement and Evaluation,1995,23(1):136
13 Peter M C, Lee E K. Striping in a RAID Level 5 Disk Array. ACM SIGMETRICS Performance Evaluation Review, 1995, 23(1):136～145
14 Yan X, Han J, Afshar R. CloSpan: Mining closed sequential patterns in large datasets. In: Proc. 2003 SIAM intConfData Mining (SDM'03),2003

(上接第 158 页)

实验中,对 Agent 的竞价行为进行仿真实验,并将实验结果与回归神经网络(RNN)及一般的强化学习算法(RL)相比较。

设 Agent$_i$ ($i=1,2,\cdots,N$)在参与某一商品交易的过程中对商品的报价是一个非平稳时间序列$\{Q_{i,j};i=1,2,\cdots,N;j=1,2,\cdots,M\}$,其中 N 为 Agent 个数,M 为竞价次数。

在实际应用中常常通过统计量来逼近模型以评价模型的性能。在本文中用均方根相对误差均值去评价算法 ANNRL。

当有 N 个 Agent 参与竞价时,为了比较基于神经网络的多 Agent 强化学习算法的性能,取 N 个 Agent l 步预测的 RMSRE 的平均值作为比较依据。其中均方根相对误差均值(the Mean of Root Mean Square Relative Error, MRMSRE)定义如下:

$$MRMSRE=\frac{1}{N}\sum_{j=1}^{N}\sqrt{\frac{1}{M}\sum_{j=1}^{m}\left(\frac{x_i-y_{j,l}}{x_i}\right)^2} \tag{19}$$

实验中取 10 个交易 Agent 对商品 U 在 157 次竞标中的报价,作为预测各交易主体在未来行动中的出价策略的数据基础。

实验结果如表 1 和图 4 所示,表中数据是分别对基于神经网络的多 Agent 强化学习算法(ANNRL)、RNN 和 RL 算法进行前 6 步预测所取得的均方根相对误差的平均值(MRMSRE)。

表 1 预测步数与均方根相对误差均值

STEP	ANNRL	RNN	RL
1	0.0015	0.0025	0.0033
2	0.0026	0.0387	0.0417
3	0.0562	0.0869	0.1169
4	0.1188	0.1321	0.2352
5	0.1297	0.2572	0.3061
6	0.1125	0.3829	0.4727

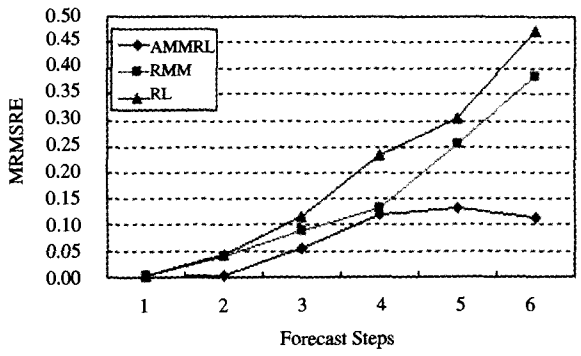

图 4 均方根相对误差均值比较效果图

实验结果表明,算法 ANNRL 在实际预测中具有非常理想的效果。在实验仿真中,仅仅 6 步预测即可获得 89.89% 的准确度,而且经过一定时间的学习后,ANNRL 算法收敛到一个极限值,这正是我们所期望的,因为每一个 Agent 是基于对其他主体和市场信息的信念而修正 Q 值的。

结论 本文设计了一个 Agent 强化学习的神经网络模型,实现了基于神经网络的 Agent 强化学习算法 ANNRL,证明了该算法在一定假设条件下收敛到最优 Q 值。由于神经网络可以逼近任意值函数,而通过值函数可以将高维空间映射到低维数据空间,因此 ANNRL 可以在一定程度上降低状态行为空间计算的 VC 维,同时通过随机梯度下降算法去最小化 Bellman 残差平方和,可以获得该算法更好的收敛速度和效果。我们在基于 Multi-Agent 的电子商务交易系统 MAECTS 中实现了 ANNRL 算法,通过对 MAECTS 中 Agent 的竞价行为进行仿真实验,验证了 ANNRL 算法具有良好的性能和行为逼近能力。

参 考 文 献

1 ZHANG Rubo, GU Guochang, et al. Reinforcement Learning Theory, Algorithm and Its Application [J]. Control Theory and Applications, 2000,(17)5: 637～641
2 Burnsa T R,Gomolinskab A. Socio-cognitive mechanisms of belief change Applications of generalized game theory to belief revision, social fabrication, and self-fulfilling prophesy [J]. Journal of Cognitive Systems Research,2001,2: 39～54
3 Kaelbling L P, Littman M L, Moore A W. Reinforcement learning: A survey [J]. Journal of Artificial Intelligence Research, 1996,4:237～285
4 Tsitsiklis J N, Roy B V. An analysis of temporal difference learning with function approximation [J]. IEEE Transactions on Automatic Control ,1997,42 (5) : 674～690
5 Watkins C J , Dayan P. Q-learning [J]. Machine Learning,1992, 8: 279～292
6 Singh S P,et al. Convergence results for single-step on policy reinforcement learning algorithms [J]. Machine Learning,2000,38 (3):287～308
7 Planning B C. Learning and coordination in multi-agent decision processes [J]. In: Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge [C]. Morgan Kaufmann,1996,195～210
8 Haykin S. NEURAL NETWORKS A Comprehensive Foundation [M]. In:Prentice Hall Tsinghua University Press, 2001
9 Sutton R S. Barto A G. Reinforcement Learning: An Introduction [M]. Cambridge, MA;MIT Press, 1998
10 ZHONG Yu,GU Guo-chang,ZHANG Ru-bo. Surey of distributed reinforcement learning algorithms in multi-agent systems [J]. Control Theory & Applications,2003,20(3):317～322
11 GAO Yang,CHEN Shi-fu,LU Xin. Research on Reinforcement Learning Technology: A Review. ACTA Automatica Sinica, 2004,30(1): 86～100
12 Taylor M E, Whiteson S, Stone P. Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain. In:Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006), Seattle, WA July 2006. 1321～1328
13 Sherstov A A,Stone P. Improving Action Selection in MDP's via Knowledge Transfer. In:Proc AAAI-2005, Pittsburgh, USA