一种自适应的基于预测的I/O性能优化方法*)

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

摘 要 目前计算机的I/O性能已经成为严重制约计算机整体性能提升的瓶颈。在现有技术条件下,I/O性能的优化方法对于提升I/O性能显得尤为重要。本文提出了一种自适应的基于预测的I/O性能的优化方法,该方法采用了基于预测的数据预读策略,改进I/O读操作的响应速度;采用基于预测的缓存分配策略,改进了I/O写操作的响应速度,从而在整体上提高了计算机系统的I/O性能。从实验测试的结果可以看出,这种方法适用于多种不同的负载类型,能够大幅度提升计算机的I/O性能,是一种通用的、效果显著的I/O性能优化方法。

关键词 自适应,基于预测,I/O性能,优化方法

A Self-adaptive and Prediction-based I/O Performance Optimization Method

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

Abstract I/O performance has already become the bottleneck of the total performance of computer systems for a long time. And under the condition of the present computer technology, I/O performance optimization method looks especially important for I/O performance. A self-adaptive and prediction-based I/O performance optimization method is proposed in this paper. The method uses prediction-based prefetching to improve the performance of read operation and uses prediction-based buffer allocation to improve the performance of write operation, so the I/O subsystem performance is enhanced totally. The results of testing experiments show that this method is universal and obviously effective to different kinds of I/O workloads.

Keywords Self-adaptive, Predication-based, I/O performance, Optimization method

1 引言

在计算机系统得到日益广泛应用的今天,计算机系统的性能也日益得到广泛的关注,用户对计算机系统的性能要求也愈来愈严格和苛刻。虽然如何提高计算机性能是一个古老的问题,但是目前显得尤为突出。

计算机系统的性能主要由CPU的处理速度和外部存储设备的I/O速度来决定。要提高计算机系统的整体性能,需要这两部分都得到充分的提高。而在过去的20年里,CPU的处理速度提高了几千倍甚至上万倍,而硬盘的I/O速度只提高了大约10倍。到目前为止,包括硬盘在内的外部存储设备的I/O性能一直是计算机系统性能的瓶颈,严重制约了计算机系统整体性能的提高。

为了提高计算机系统的I/O性能,众多研究人员进行了广泛和深入的研究,提出了一些卓有成效的I/O性能优化方法。总的来说这些方法可以分为以下三类:

• 使用数据缓存。使得一部分I/O操作只需要访问内存就足够了,而不需要访问外部存储设备,这样就会减少I/O操作对外部存储设备的访问,从而提高I/O操作的平均响应速度;

• 调度I/O操作。将数据地址邻近的I/O操作放在一起执行,以减少硬盘的寻道时间,充分利用硬盘的数据带宽;

• 数据预读策略。将后来可能使用的数据预先读入内存中,后续I/O操作中的一部分只需要访问内存而不需要访问外部存储设备,从而提高了计算机系统的平均I/O响应速度。

其中,数据缓存的使用是提高I/O性能的一个里程碑,它革命性地缩短了平均I/O响应时间;而I/O调度和预读策略是技术性的改进,在一定程度上也提高了计算机系统的I/O性能,有的甚至是显著地改进了I/O性能。

然而,目前的I/O性能优化方法,不管它属于以上三类中的哪一类,都是针对某一种固定的I/O负载类型设计的,都有一个普遍的缺陷:缺乏自适应性,无法适应其他负载类型。所谓的自适应性,就是优化方法可以根据I/O负载特征的变化而不断调整自身的策略,使得优化方法能够动态地、自动地与负载特征相适应,始终做到对当前I/O性能的最优化调节。

本文针对现有I/O性能优化方法的上述不足,提出了一种自适应的基于预测的I/O性能优化方法。对于I/O请求中的读操作,该方法首先是用历史数据预测下一个读操作的逻辑块地址和预读数据量,然后执行预读操作;对于I/O请求中的写操作,该方法预测所需要的缓存数量,然后分配一块相应大小的写缓存供写请求使用。本文以下部分的内容如下:第2部分讨论这种自适应的基于预测的I/O性能优化方法;第3部分说明这种方法的实现;第4部分实验测试和分析这种方法的性能;第5部分给出了2000年以来国内外的相关研究;

*)国家CNGI产业化类项目(分项CNGI-04-5-1D)资助。严 琪 硕士研究生,主要研究方向为网络存储、射频识别;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;胡庆成 硕士研究生,主要研究方向是数据存储;李益民 硕士研究生,主要研究方向是数据存储。

最后是总结。

2 自适应的基于预测的I/O性能优化方法

2.1 方法描述

计算机系统中的I/O操作可以划分为读操作和写操作两种。虽然控制操作在功能上相对独立，但是可以最终归属于读操作或者写操作。读操作和写操作有不同的特征，为了优化方法的准确性，不宜采用相同的策略进行优化，所以在本文所述的优化方法中，对读操作和写操作区分对待。方法的整体描述如下：

(1)分别为读操作和写操作分配各自的数据缓存。读操作缓存的大小是固定的，一般是128kB，写操作缓存的大小随着预测算法动态变化；

(2)对于读操作，依据数据访问地址和访问数据量的历史数据，预测下一次要访问的数据地址和访问数据量，预先从外部存储设备将相关数据读入读操作缓存，缓存的替代算法使用LRU(Least Recently Used)算法；

(3)对于数据读操作，首先到读操作缓存寻找。如果没有命中，再到写操作缓存寻找。如果还未命中，最后访问外部存储设备；

(4)对于写操作，依据请求数据量的历史数据，估算下一次写操作的数据量，然后预测下一次写操作所需的数据缓存量，最后分配相应大小的写操作缓存；

(5)对于数据写操作，如果有空闲的写操作缓存，就将数据写入缓存。否则先将写操作缓存中的数据写入外部存储设备，然后将写操作的数据写入写操作缓存。

(6)读操作缓存只能保存读操作的预读数据，不能被写操作写入。而写操作的数据缓存只能被写操作更新，不能保存预读数据。

纵观整个方法的步骤，最关键的也是最困难的就是第2步和第4步的I/O读操作和写操作的预测及优化，其余步骤都是水到渠成，顺理成章的。下面将对第2步和第4步进行详细说明。

2.2 I/O读操作的预测和优化

I/O读操作的预测包括两个方面：第一是对预读数据地址的预测，第二是对预读数据量的预测。

预读数据地址的预测方法包括以下步骤：

(1)从计算机启动开始。当存储设备驱动程序向存储设备控制器第 j 次发送I/O读命令时，记录该I/O读命令的逻辑块地址A[j]和I/O读命令的发送时间T[j]，计算并记录逻辑块地址增量D[j]，D[j]=A[j]−A[$j-1$]且A[0]=0。使用A[j]、T[j]、D[j]构成一个记录项，将这个记录项插入一个记录链表。

(2)对记录链表中的逻辑块地址增量进行预处理。具体包括如下三个子步骤：

1)计算DL=max{D[j]}− min{D[j]}。如果DL小于常数阈值32，则不对记录链表做任何处理，否则转入下一子步骤；

2)如果DL大于常数阈值32，计算记录链表中所有D[j]的标准差SD。如果标准差SD小于常数阈值40，则不对记录链表做任何处理，否则转入下一子步骤；

3)如果标准差SD大于常数阈值40，将记录链表中所有记录项的D[j]的均值记为ave{D[j]}，标记记录链表中D[j]与ave{D[j]}之间的距离大于40的记录项。如果所有的记录项都被标记，则去除T[j]距离当前时间最近的5～10个记录项的标记。最后删除记录链表中被标记的记录项。

(3)计算预读地址增量。预读地址增量等于预处理后的记录链表中所有记录项的逻辑块地址增量D[j]的均值。

(4)计算预读数据地址。预读数据地址等于最近一次I/O读命令的逻辑块地址与预读地址增量之和。

预读数据量的预测方法包括以下步骤：

(1)从计算机启动开始，到达存储设备驱动程序的第 i 个读请求的请求数据量记为S[i]；预读数据量记为PS；读操作缓存大小固定，记为BS。

(2)当第 i 个读请求到达驱动程序时，计算预读数据量PS，PS=min{0.2PS+0.8S[i]，0.3BS}，系统启动时PS=0。

(3)当第 $i+1$ 个读请求到达驱动程序时，使用上一步的方法重新计算预读数据量。

I/O读操作的优化是依据上面计算出的预读数据地址和预读数据量，从外部存储设备将相应的数据读入读操作缓存。为了避免优化方法的复杂性，缓存内容的替代策略使用LRU，即读操作缓存中最早被使用的数据被预读数据覆盖。I/O读操作的优化使得大部分来自上层应用程序的I/O读操作只需要从系统内存读取数据，无需直接访问外部存储设备，从而大大加快了面向应用程序的读请求响应速度。

2.3 I/O写操作的预测和优化

I/O写操作的预测包括下一个写请求的请求数据量的预测，预测方法包括以下步骤：

(1)从计算机启动开始，到达存储设备驱动程序的第 i 个写请求的请求数据量记为S[i]；系统内存总量记为MS；写请求的数据量记为WS。

(2)当第 i 个写请求到达驱动程序后，估算第 $i+1$ 个写请求的数据量WS，WS=min{0.2WS+0.8S[i]，0.1MS}，WS的初始值为0。

(3)当第 $i+1$ 个写请求到达驱动程序时，使用上一步的方法重新计算第 $i+2$ 个写请求的请求数据量。

I/O写操作的优化是指：在估算出下一个写请求的数据量以后，驱动程序在下一个写请求到达之前为其分配相应大小的、独立的写操作缓存，供其使用。当计算机系统较为空闲时，或者系统内存短缺时，再将缓存数据写入外部存储设备，以释放系统内存。这样做的目的是大部分来自应用程序的写操作只需要将数据写入内存，而无需直接访问外部存储设备，从而大大加快面向应用程序的写请求响应速度。

2.4 方法总结

这种自适应的基于预测的I/O性能优化方法采用了基于预测的数据预读策略，改进I/O读操作的响应速度；采用基于预测缓存分配策略，改进了I/O写操作的响应速度，从而在整体上提高了计算机系统的I/O性能。对于计算机系统整体性能的提高是非常有意义的。

3 实现方法说明

一般来说，I/O性能优化方法主要在驱动程序层和存储设备层实现。这两者相比较而言，驱动程序层的实现更加灵活，而且可以借助主机的处理资源和内存资源运行比较复杂的优化算法，但是系统开销会比较大；存储设备层的实现受可用资源的限制，缺乏灵活性，且只能运行相对简单的优化算法，但是系统开销小。

这种自适应的基于预测的I/O性能优化方法有一定量的

计算开销，并且需要大量系统内存做支持，所以在驱动程序层实现比较合理。并且，处理器运算速度的迅速提高和内存容量的快速上升为这种优化方法在驱动程序层的实现提供了广阔的空间。

驱动程序层的实现通过对外部存储设备驱动程序做一定修改，在驱动程序源码的合适位置插入优化方法的实现代码来完成。对加入优化方法代码的驱动程序源码进行编译，就可以获得融合了优化方法的驱动程序。下面将对未加优化方法的外部存储设备驱动程序称为“原驱动程序”，将加入本文所述方法的驱动程序称为“优化的驱动程序”。下面的实验测试就是测试在不同类型的负载下、在原驱动程序和优化的驱动程序分别运行时计算机系统的 I/O 性能。

4 实验测试与分析

4.1 测试负载和测试方法

为了测试这种自适应的基于预测的 I/O 性能优化方法的有效性和适用性，使用了三种负载类型进行测试。这三种负载类型分别是文件服务器负载、Web 服务器负载和数据库服务器负载。为了测试该方法在真实负载下的表现，文件服务器负载采用清华大学信研院 Web 与软件研发中心的共享文件服务器负载；Web 服务器负载选用清华大学信研院 Web 与软件研发中心维护的数字资源管理系统（DRMS）负载；数据库服务器负载采用清华大学信研院 Web 与软件研发中心维护宏观经济库应用负载。

测试方法简要描述如下：

（1）在原驱动程序中加入 I/O 性能数据采集模块后，将其运行于上述三种负载之下，获得原驱动程序的 I/O 请求响应时间数据；

（2）在优化的驱动程序中加入 I/O 性能数据采集模块后，将其运行于上述三种负载之下，获得优化的驱动程序的 I/O 请求响应时间数据；

（3）对上面得到的两组数据进行对比分析。

下面分别对文件服务器负载、Web 服务器负载和数据库服务器负载下的 I/O 性能的测试结果进行展示和分析。

4.2 文件服务器负载测试数据

图 1 是文件服务器负载下外部存储设备驱动程序对读请求的响应时间，横坐标是读请求的数据量，纵坐标是读请求的平均响应时间。

图 2 是文件服务器负载下外部存储设备驱动程序对写请求的响应时间，横坐标是写请求的数据量，纵坐标是写请求的平均响应时间。

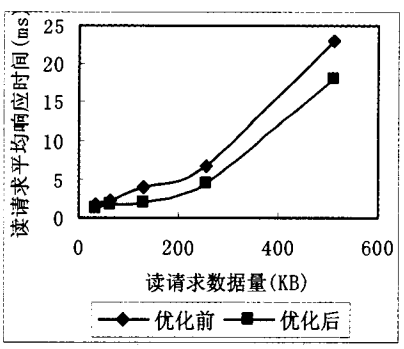

图 1 读请求响应时间

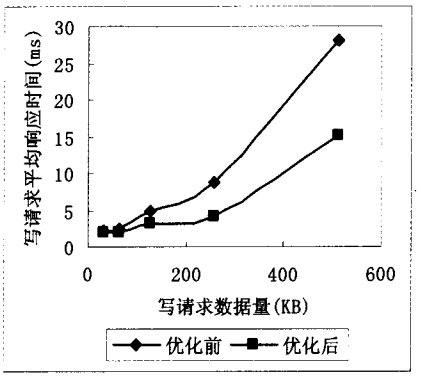

图 2 写请求响应时间

4.3 Web 服务器负载测试数据

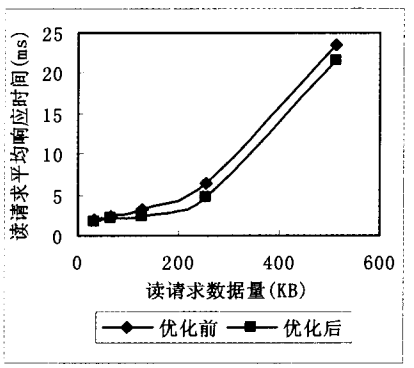

图 3 读请求响应时间

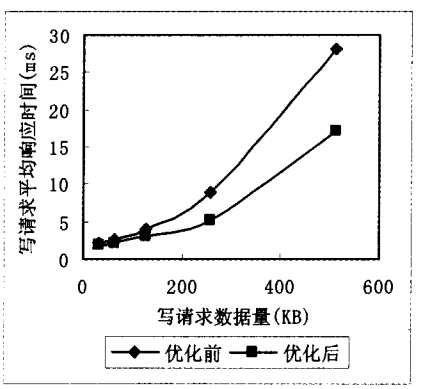

图 4 写请求响应时间

图 3 是 Web 服务器负载下外部存储设备驱动程序对读请求的响应时间，横坐标是读请求的数据量，纵坐标是读请求的平均响应时间。

图 4 是 Web 服务器负载下外部存储设备驱动程序对写请求的响应时间，横坐标是写请求的数据量，纵坐标是写请求的平均响应时间。

4.4 数据库服务器负载测试数据

图 5 是数据库服务器负载下外部存储设备驱动程序对读请求的响应时间，横坐标是读请求的数据量，纵坐标是读请求的平均响应时间。

图 6 是数据库服务器负载下外部存储设备驱动程序对写请求的响应时间，横坐标是写请求的数据量，纵坐标是写请求的平均响应时间。

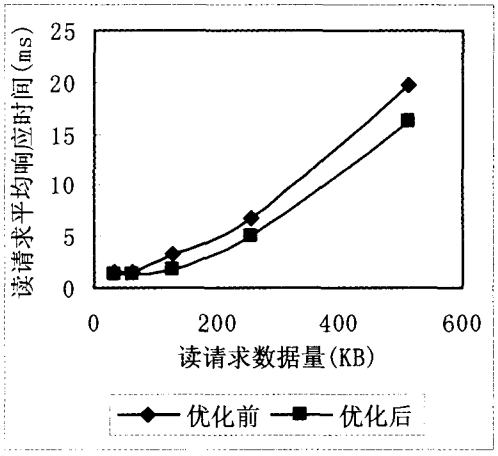

图 5　读请求响应时间

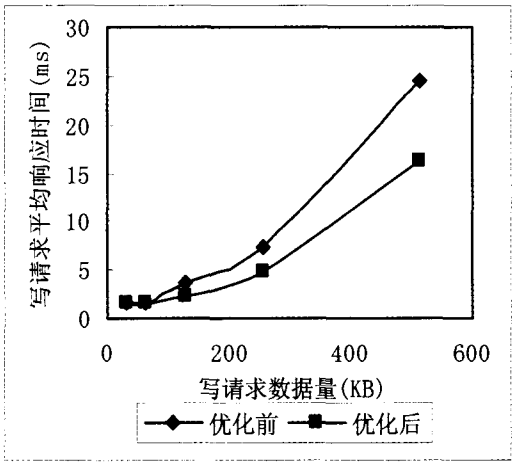

图 6　写请求响应时间

4.5　I/O 响应时间测试数据分析

从三种不同类型的真实负载下的测试数据可以看出，在优化方法采用之前，不论对于读操作还是写操作，当请求的数据量小于等于 64kB 时，I/O 响应时间都比较小。而且随着数据的增大，I/O 响应时间的上升也较慢。当请求的数据量大于等于 64kB 时，随着请求数据量的增长，I/O 响应时间迅速上升。这是因为原驱动程序维护 64kB 大小的缓存，当请求数据量小于缓存尺寸时，相当一部分 I/O 操作只对内存进行，没有直接访问外部存储设备。又由于访问内存的时间远远小于访问外设的时间，所以响应时间自然比较小；而当请求数据量大于缓存尺寸时，缓存已不足以容纳所请求的数据量，I/O 操作需要直接访问外设来获取数据，访问外部存储设备的开销使得 I/O 响应时间随数据量上升而迅速上涨。

对于读操作，优化方法在请求数据量小于 64kB 时对 I/O 响应时间的改进非常小，这是因为未优化之前的数据缓存已经得到了很好的性能，而优化方法只能得到类似的性能；当请求数据量大于 64kB 以后，一方面由于优化方法采用更大的缓存(128kB)，另一方面由于数据的预读策略使得相当一部分读请求只从内存读取数据，省掉了大量的访问外设的时间，因而带来很可观的 I/O 性能的改进。这也说明预读算法本身是比较有效的，能够比较准确地预测后续访问所需要的数据。

对于写操作，优化方法在请求数据量小于 64kB 时对 I/O 响应时间的改进同样非常小，这也是因为未优化之前的数据缓存已经得到了很好的性能，而优化方法只能得到类似的性能；但是当请求数据量大于 64kB 时，在优化之前，一部分数据甚至大部分数据需要直接写入外部存储设备，而优化方法按照下一个写请求的预测数据量分配数据缓存，对于请求数据量不是很大的写请求可以做到“按需分配”，这使得大多数的写请求只需要和内存打交道而无需直接写入外部存储设备，因而大大缩短了写请求的 I/O 响应时间。

从图 1 到图 6 可以看出，本文所述方法对于请求 I/O 响应时间的改进好于对读请求 I/O 响应时间的改进，这是因为：读操作的优化方法是固定缓存大小的预读策略，预读策略的准确性不够完备，预读失误时会带来访问外存的开销；而写操作的优化采用“按需分配”缓存的原则，虽然内存分配与释放会有一些开销，但却大大减少了访问外存的次数，利远远大于弊。

从图 1 到图 6 也可以看出，本文所述方法是一种比较通用的 I/O 性能优化方法，它对三种不同类型的负载的 I/O 性能都有显著的改进。但是，本方法需要计算机系统拥有富余的处理能力和足够的内存可供使用。如果系统缺乏处理能力或者内存短缺，这种优化方法将收效甚微，甚至会适得其反。在采用任何 I/O 性能优化算法之前都要考虑其应用条件，本文所述方法的使用也不例外。

5　相关研究

I/O 性能一直是计算机性能的瓶颈，I/O 性能的优化问题一直得到科学技术人员的广泛关注。从 2000 年到现在，I/O 性能的优化问题仍旧是计算机科学方面的热点问题。下面就国内外这方面的研究做一简单介绍。

国内的研究主要集中在磁盘阵列和网络存储技术的 I/O 性能：文[1]研究了不同负载类型下存储网络的性能；文[2]给出了一种聚合 I/O 命令以提高磁盘阵列 I/O 性能的方法；文[3]研究了事务处理负载下的磁盘阵列性能；文[6] 通过优化 I/O 操作的执行顺序和对数据分块提高 I/O 操作的并行性来提高外部存储设备的 I/O 性能。

国外的研究范围比较广泛，几乎每个与 I/O 性能相关的技术点都有涉及：文[4]通过实验的方式研究了不同应用的 I/O 优化的价值，发现不同的优化方法适用于不同的应用场景；文[5]通过研究 I/O 负载的特征发现 I/O 负载具有脉冲特性和自相似特性，建议 I/O 性能需要优化并且又得到优化的余地；文[7]建议将数据量很小的活跃数据保存在缓存中以及探索读操作的顺序性并按照这种顺序预读数据来提高 I/O 性能；文[8]的研究证明数据顺序预读和写缓存的使用对于 I/O 性能的提高意义重大，建议用外存容量 1%的内存充当数据缓存；文[9]给出了一种同时考虑时间和空间局部性的读缓存管理方法；文[10]给出了一种同时考虑时间和空间局部性的写缓存管理方法；文[11] 通过应用数据挖掘的方法，建立了一种计算存储设备的数据块相关性的算法，并使用数据块相关性指导数据预读和数据分布。

总结　本文给出了一种自适应的基于预测的 I/O 性能优化方法，该方法对读操作和写操作分别进行优化。对于读操作，该方法使用历史上的读操作的逻辑块地址数据预测下一次读操作的逻辑块地址，使用历史上的读请求的请求数据量预测下一次读请求的请求数据量，然后执行预读操作。对于写操作，该方法使用历史上的写请求的请求数据量预测下一

(下转封三)

示供电区域内的地理实体，分别用一定的代码及一对或一系列坐标予以描述。点是不依赖比例尺的独立符号，可表示设备如开关、变压器、杆塔等；线是一组有序点的组合，表示所有线状物体，如道路、围墙、架空线及电缆等；面是封闭的弧线，可表示配电系统供电区域等。在GIS中的每个点、线、面元素的属性中均包含拓扑连接关系。构建电力生产管理系统的拓扑数据方法如下：

首先读取基本的点、线信息，构建设备和线路的拓扑连接关系。我们通过MapObjects提供的一套数据访问对象对存储在.shp文件中的地理数据进行读写。由于.shp文件存放了几何实体的地理信息，所以通过读取设备层文件（如Device.shp）就可以得到所有设备的地理信息；读取线路层（如Line.shp）则可以得到所有线路的地理信息，并建立线路与设备的拓扑连接关系。然后建立符合电力应用的拓扑数据。由于GIS对站内、站外设备采用不同方式进行管理。站内设备和线路用主站接线图表示，站外所有设备和线路则用地理图表示。主站在地理图上仅表示为一个点，只有在查询主站详细情况时才调用主站接线图。所以，要构建电力拓扑数据，需要分别对这两类图进行处理。对于主站接线图，GIS中的母线是由线段构成，而电力拓扑数据定义母线为1个结点。其处理方法是：通过搜索GIS主站接线图，将母线进行合并，得到母线信息。再进一步搜索，得到站内所有设备和线路的连接信息，同时将站内的线路标识为零阻抗线。

定义其数据结构如下：

```
Seg_node=record//分段结点信息数据结构
Node_no:integer;//结点序号
Node_InID:string;//结点本身的标识码
Node_type:string;//分段结点的类型：如分支分段结点、开关分支结点、导线规格不同的分段结点
Node_pos:imopoint;//结点坐标
Edg_list:pSegedge;//以此结点为起点的段的列表
end;
Gen_node=record//母线结点信息数据结构
Gen_ArcList:Tlist;//构成本母线的各个弧的列表
Seg_nodeID:integer;//母线结点编码
end;
```

对于站外地理图，一般首先进行分段。分段是指给定分段条件，把满足给定条件的基本线路作为同一个基本段。一般分段条件包括：按厂站、开关、负荷点、分支点（非设备的结点）分段。其处理的方法是：对站外地理图进行搜索，当搜索到厂站、开关、负荷点及分支点时，则作为新段的开始，否则将搜索到的基本线路进行合并，得到段信息。在此过程中，记录搜索到的设备信息，设备作为线路的端点。在GIS中，由于符号化和制图的需要，所有电力设备的处理均为1个点，而电力设备有单端元件和双端元件之分，所以，还需对诸如开关等双端元件做进一步处理，使得开关在GIS图上表现为1个点，而在逻辑上分割为双端元件。定义其数据结构如下：

```
pSegedge=^segedge//指向段记录的指针
Seg_edge=record//段信息数据结构，段是满足一定条件的弧的集合
Line_list:string;//构成本段的各弧的列表
From_nodeID:integer;//段的起始结点号
To_nodeID:integer;//段的终点结点号
end;
```

利用以上定义的数据结构，可以生成配电管理高级应用所需的数据。

结论与展望 本文提出了以“空间数据为载体、以设备管理为核心、以业务规划为灵魂、以流程重组为主线、以生产管理为目标”的电力生产管理系统方案，该系统利用计算机、网络技术，将供电生产管理中的输电网络、变电站所、中低压配电网络等的分布、属性及实时信息按实际地理位置描述在地理背景图上，集查询统计、运行维护、生产业务管理、应用分析、辅助决策等功能于一体。通过该系统，可以便捷地了解电网分布、相关属性及实时信息，并支持业扩、事故抢修、可靠性分析等管理过程，以提高效益、降低成本、提升电网运行管理水平，为企业进一步适应市场化经营和提供高品质服务奠定基础。

参考文献

1 许中平，张军伟. WebGIS技术的实现及在电力系统中的应用[J]. 遥感技术与应用，2002，17(4)

2 黄芳，吴元龙. GIS在电力信息系统中的应用[J]. 计算机工程，2001，27(5)

3 姚翔，陈林，董凤宇. 城市中电缆进线GIS变电站核相研究[J]. 高电压技术，2005，31(2)：81～84

4 刘莹，米建华，等. 电网地理信息系统之WEB-GIS与高级应用功能设计[J]. 电力信息化，2004，2(4)：32～35

5 朱陶业，孙喜梅，等. 几种流行网络地理信息系统的模式比较研究[J]. 计算机工程与应用，2002，38(15)：77～83

（上接第268页）

次写请求的数据量，依据写请求的数据量的预测值动态地分配写请求所需的缓存。本文所述方法本质上是通过智能的缓存来减少I/O操作对外部设备的访问，以提高面向应用的I/O响应速度。

不同类型的真实负载对这种自适应优化方法的实验测试结果说明，这种方法能够适应多种不同的负载类型，并且在I/O性能优化方面有显著效果，可以说这是一种比较通用的I/O性能优化方法，从而克服了传统I/O性能优化方法适用负载类型单一的缺陷。

参考文献

1 曹强，谢长生. 网络存储系统中I/O请求响应时间的研究. 计算机研究与发展，2003，40(8)：1271～1276

2 陈琼，张江陵，冯丹. 一种提高磁盘阵列I/O性能的策略[J]. 小型微型计算机系统，2000，21(1)：13～15

3 冯丹，陈琼，张江陵. 事务处理环境下的磁盘阵列响应时间分析. 小型微型计算机系统，2001，22(4)：401～404

4 Kandaswamy MA, Kandemir M, Choudhary A, et al. An experimental evaluation of I/O optimizations on different applications. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(12): 1303～1319

5 Hsu W W, Smith A J. Characteristics of I/O traffic in personal computer and server workloads[J]. IBM Systems Journal, 2003, 42(2): 347～372

6 Di W, Shu JW, Shen MM. Data I/O optimization in storage systems. Lecture Notes in Computer Science, 2004, 3252: 294～302

7 Hsu WW, Smith AJ, Young HC. The automatic improvement of locality in storage systems. ACM Transactions on Computer Systems, 2005, 23(4): 424～473

8 Hsu W W, Smith A J. The performance impact of I/O optimizations and disk improvements. IBM Journal of Research and Development, 2004, 48(2): 255～289

9 Jiang Song, Ding Xiaoning, Chen Feng, et al. DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Locality. In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST 2005), San Francisco, CA, December 2005. 257～270

10 Gill B S, Modha D S. WOW: Wise Ordering for Writes-Combining Spatial and Temporal Locality in Non-Volatile Caches. In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST 2005), San Francisco, CA, December 2005. 165～183

11 Li Zhenmin, Chen Zhifeng, Sudarshan M, et al. C-Miner: Mining Block Correlations in Storage Systems. In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST 2004). San Francisco, CA, March 2004. 212～230