ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, Vol.17, No.11, November 2006, pp.2234−2244
DOI: 10.1360/jos172234

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# 一种内存网格的数据预取算法*

褚 瑞+, 卢锡城, 肖 侬

(并行与分布处理国家重点实验室,湖南 长沙 410073)

## A Data Prefetching Algorithm for RAM Grid

CHU Rui+, LU Xi-Cheng, XIAO Nong

(National Laboratory for Parallel and Distributed Processing, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4575835, Fax: +86-731-4575835, E-mail: rchu@nudt.edu.cn, http://www.nudt.edu.cn

**Chu R, Lu XC, Xiao N. A data prefetching algorithm for RAM grid. *Journal of Software*, 2006,17(11): 2234−2244.** http://www.jos.org.cn/1000-9825/17/2234.htm

**Abstract**: RAM (random access memory) Grid is a new grid system aiming at memory resources sharing in wide-area network; it can improve the performance of memory intensive or IO intensive applications when lack of physical memory. Reducing or hiding the network overhead can improve the performance of RAM Grid, which lies on the overhead of network communication. In this paper, through the analysis of RAM Grid, a "push" based prefetching mechanism is proposed for it. The corresponding prefetching algorithm, which comes from the sequential pattern mining method in data mining area, is also raised. The prefetching algorithm is evaluated and proved by trace driven simulation.

**Key words**: RAM (random access memory) grid; prefetching algorithm; sequential patterns mining; IO intensive; disk cache

摘 要: 内存网格(RAM(random access memory) grid)是一种面向广域网上内存资源共享的新型网格系统.它的主要目标是在物理内存不足的情况下,提高内存密集型应用或 IO 密集型应用的系统性能.内存网格的应用效果取决于网络通信开销.在减少或隐藏网络通信开销的情况下,其性能可以进一步提高.通过对内存网格的分析,设计了一种基于"推"数据的内存网格预取机制.借助数据挖掘领域中序列模式挖掘的方法,提出了相应的预取算法.通过基于真实运行状态的模拟,对预取算法进行了评估和验证.

关键词: 内存网格;预取算法;序列模式挖掘;IO 密集型;磁盘缓存
中图法分类号: TP393 文献标识码: A

随着网格计算技术在各个领域的不断应用和发展,网格的概念也正在发生多方面的延伸.在传统意义上,可以把网格分为计算网格和数据网格两大类[1−4],分别用于广域网范围内计算资源的共享和数据资源、海量存储资源的共享.作为计算机系统中最重要的资源之一,内存(random access memory,简称 RAM)资源也需要借助网

* Supported by the National Natural Science Foundation of China under Grant Nos.60573135, 60673167, 90412011 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant Nos.2003CB317008, 2005CB321801 (国家重点基础研究发展规划(973))

Received 2006-06-10; Accepted 2006-08-07

格技术,实现面向广域网络的共享.在我们之前的研究工作中[5],已经结合 20 世纪 90 年代流行的网络内存技术[6−8]和目前的面向服务计算[9]、网格计算[1,4]等技术,提出了内存网格(RAM grid)的概念.内存网格是计算网格和数据网格的扩充,其主要目标是利用高速互联的广域网络上分布的大量空闲内存资源,实现一种分布式的内存共享系统,解决部分节点物理内存不足的问题.与面向集群的网络内存技术相比,内存网格的分布范围更为广泛,涉及的空闲资源更多,同时,针对节点的动态性、异构性和自治性进行了重点优化设计.

内存网格的应用范围主要包括两个方面:首先,对于在运行过程中需要占用大量内存,甚至内存需求量超过了宿主节点物理内存容量的“内存密集型”应用,如科学计算中的核模拟、气象模拟应用以及我们在研究工作中经常使用的网络模拟器等.内存网格借助操作系统中的虚拟内存和换页技术,把一部分暂时不用的内存页帧换入远程的空闲内存中,在必要时再取回这些页帧.与传统的磁盘换页机制相比,通过高速网络互联的内存资源能够提供更快的换页速度,从而提高系统性能.另外,对于运行过程中需要产生大量小粒度、非连续磁盘 IO 操作的应用,如 Web 服务器、大型数据库系统等等,内存网格能够利用广域网上大量的内存资源,实现对 IO 操作的缓存.特别是在物理内存不足的情况下,系统本身的磁盘缓存机制几乎失效,而广域网上的空闲内存资源往往比较丰富,对 IO 密集型应用的缓存效果也更为明显.

不难看出,无论是上述哪种应用,内存网格的关键都在于利用高速互联的广域网络上的内存资源,填补目前计算机中内存和磁盘之间的巨大性能差异.仅从访问延迟来看,根据 Patterson 等人的研究[10],目前硬盘的延迟大约在 5ms~35ms,内存的延迟大约在 52ns~225ns,两者之间的差距高达 5~6 个数量级.广域网的延迟通常存在较大的差异,从数百微秒到数秒不等.但目前常用的校园网、园区网或城域网往往都具有较低的延迟,根据实际测试,校园网内部节点间的网络延迟均低于 1ms,而在同一城市的节点间,网络延迟也大都不超过 3ms.如果能有效地利用这些高速广域网络上的空闲内存资源,则完全可能在目前的网络环境中构造出实用化的内存网格系统,从而提高系统的性能.

同时,我们也可以看出,内存网格的应用效果完全取决于广域网络的通信开销.如果能减少或隐藏网络通信开销,则内存网格的性能还可以进一步提升.数据预取(prefetching)是计算机系统中最常见的隐藏通信或传输开销的技术之一,在各个层面上都有成功的应用,比如:在 CPU 片内通过硬件实现预取,隐藏二级 cache 或内存的访问开销[11−13];操作系统在磁盘读取过程中进行预读(read ahead),隐藏磁盘访问的开销;Web 浏览器对相关 Web 页面进行预读,隐藏网络访问的开销[14,15].由于内存网格本身的特性,使得它在实现数据预取的过程中,面临的背景在以下几个方面存在不同:

- 在内存网格中,数据预取的对象是远程的内存资源.与本地内存或磁盘不同,远程内存资源不仅是一种被动的存储器,也具有一定的计算能力.如果由远程内存资源进行关于预取的计算工作,则可以避免内存网格用户执行预取算法的计算开销,也可以实现更为复杂的预取算法.相比之下,大多数系统中为回避预取带来的额外计算开销,其预取算法相对比较简单.
- 由远程内存资源进行预取的计算工作,使得内存网格用户不必发出显式的预取命令,而是由内存网格把可能需要的数据“推”向内存消费者,从而进一步减少了通信开销,降低了内存消费者的设计复杂度.相比之下,目前多种 CPU 都设计了专门的预取指令并需要编译器的支持,操作系统中也通过显式插入异步读操作实现磁盘数据预取,不仅系统复杂度高,也具有较大的额外开销.
- 使用内存网格的节点往往具有本地内存不足的特点,而数据预取必须耗费一定的本地内存.因此,需要对内存网格中数据预取的过程进行分析和优化,减少本地内存的消耗.

针对上述特点,我们借鉴数据挖掘领域中比较成熟的序列模式挖掘(sequential patterns mining)技术[16],提出了一种用于内存网格的数据预取算法.本文的贡献主要包括以下几个方面:

- 我们在内存网格的工作基础上,提出了基于“推”数据的预取技术.与其他预取方法相比,其性能更高,开销更少.
- 通过分析内存网格的特点,我们借助数据挖掘技术,提出了一种新的数据预取算法,进一步提高了内存网格的性能.

- 通过基于真实运行状态的模拟,我们对内存网格中数据预取算法的性能进行了评价和验证.

本文第 1 节介绍相关研究工作.第 2 节分析内存网格中进行数据预取的一些关键问题.第 3 节具体提出一种内存网格的数据预取算法.第 4 节进行模拟实现并分析其结果.第 5 节总结当前工作.

## 1 相关工作

内存网格的概念源于 20 世纪 90 年代流行的“网络内存”技术[6−8].典型的网络内存系统可以按照应用范围的不同分为两类:一类是以提供远程换页设备、提高内存密集型应用的性能为目标的,如 Feeley 等人提出的被称为 GMS (global memory service)的全局内存管理系统[6],是这方面最经典的系统之一.此外,还包括 Markatos 等人研究的一种类似于 GMS 的网络内存系统,称为 Remote Memory Pager[7]以及 Acharya 等人以系统 API(application programming interface)的形式实现的一种名为 Dodo 的远程内存资源管理系统[8]等等;另一类是使用网络内存实现本地磁盘的缓存[17],或者高速的网络内存文件系统[18].如 Flouris 等人提出的 Network RamDisk 系统[19]以及 Anderson 等人研究的网络文件系统[18],等等.网络内存的研究工作通常是在集群内部实现的,其性能对整个集群的负载和内部的网络拥塞非常敏感[20].而内存网格则关注广域网范围内大量异构、自治的内存资源的共享,有效地弥补了网络内存的不足,应用范围更广.

在 GMS 的研究工作中,Voelker 等人对网络内存中的预取和缓冲技术进行了较为深入的探讨[21].事实上,预取技术在计算机系统中的应用非常广泛.除了操作系统中常见的磁盘预读之外,Chilimbi,Wenisch,Shi 等人的工作分别涉及到了 CPU 片内的预取技术[11−13];Swaminathan 和 Yu 等人则分别通过对用户行为的分析,实现了对 Web 页面的预取[14,15];Yang 等人提出的面向“推”数据的预取策略[22]在传统的预取方法上有了新的拓展.我们通过借鉴上述预取算法和策略的优点,提出了一种适应内存网格特性的预取算法.

我们的预取算法结合了数据挖掘领域的序列模式挖掘技术.序列模式挖掘的概念首次由 Rakesh 等人提出[16],他们的贡献还在于提出了基于演绎的序列模式挖掘算法[23];Ayres 等人提出的 SPAM 算法在 Rakesh 等人工作的基础上作了改进[24],主要是采用深度优先的搜索策略和位图表示方法;针对一般的序列模式挖掘算法在面对超长序列时性能不佳的缺点,Pei 等人提出了 PrefixSpan 算法[25].我们的算法主要借鉴 PrefixSpan,但针对问题背景的不同,对算法进行了修改,并对其正确性进行了分析和证明.

## 2 系统分析

### 2.1 概 述

为了更好地适应广域网上内存资源的异构、自治、动态等特性,我们把内存网格中的节点分为 5 种不同的类型,分别称为空闲节点、代理节点、内存节点、中间节点和用户节点[5].空闲节点是指广域网上具有一定空闲 CPU 资源和内存资源,可以对外提供服务的节点;内存节点是指正在对外提供远程内存的节点;中间节点是指本地资源占用较多,无法对外提供服务,也没有使用服务的节点;用户节点是指本地物理内存严重不足,需要使用远程内存的节点;代理节点是指一类特殊的内存节点,它能够把空闲节点组织起来,替用户节点查找和分配空闲节点.

随着节点本身的特性发生变化,节点可能在这 5 种类型之间发生转换,如图 1 所示.当空闲节点被分配给某个用户节点开始使用后,空闲节点转换为内存节点,如图中①所示;相反地,当空闲节点自身内存占用率上升,无法对外提供服务时,它将转换为中间节点,如②所示;随着中间节点的内存占用继续上升,需要使用远程内存,则转换为用户节点,如③所示;空闲节点也可能被选作代理节点,以组织其他空闲节点,如④所示.注意:上述 4 种转换都有相应的逆过程,具体不再赘述.代理节点和内存节点在提供服务的过程中,可能由于其自治性导致本地资源占用上升,无法继续提供服务,这时将进行服务失败的处理,并转换为中间节点,如⑤,⑥所示.

在这 5 种类型的节点中,对整个内存网格的性能影响最大的是内存节点和用户节点.这两种节点的基本功能是:用户节点把一定数量的页帧(page frame)交给内存节点保存,并在必要时取回这些页帧.为叙述方便,我们

在后文中把上述操作分别称为用户节点的“写”和“读”操作,并以页帧作为这两种操作的基本单位.

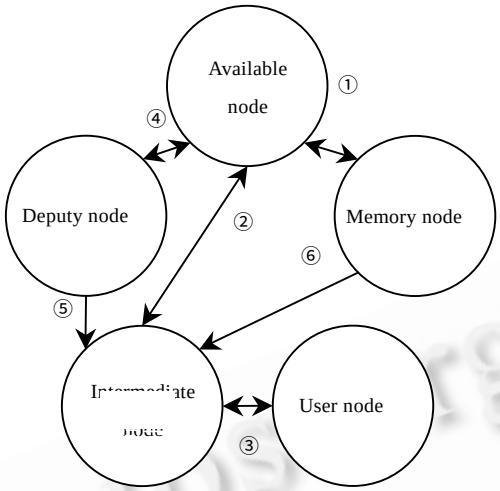


Fig.1 Nodes and their transitions in RAM grid

图1 内存网格中的节点及其转换关系

页帧在广域网络上的传输效率直接关系到内存网格的实用性.相对来说,写操作还可以异步进行,而在进行读操作时,则必须暂停当前工作,待需要的页帧传输完毕后才能继续.这一暂停等待的过程将耗用较多的时间.为使关注的问题集中化,后文以内存网格在密集 IO 中的缓存这一应用为例,集中考虑用户节点在读操作过程中的细节.在这类应用中,内存网格实际上是作为系统中的第二级磁盘缓存出现的,类似于 CPU 和内存之间的多级缓存机制,如图 2 所示.

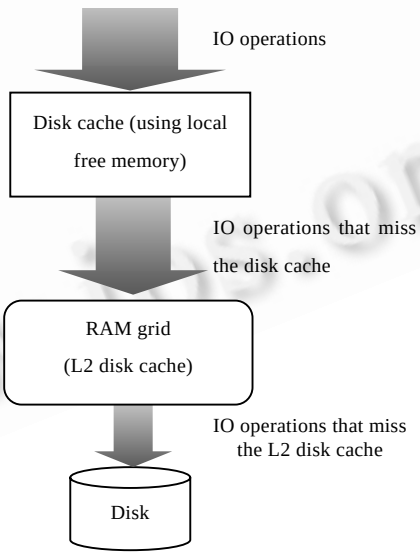


Fig.2 The effect of RAM grid in IO intensive applications

图 2 内存网格在 IO 密集型应用中的作用

由图 2 可见,内存网格中保存的每个页帧实际上都是用户节点上某个磁盘块的缓存.也就是说,内存网格中的页帧总有一个或多个对应的镜像磁盘块.因此,我们在后文中将不再严格区分“页帧”和“磁盘块”以及“页帧号”和“磁盘块号”之间的不同.

### 2.2 预取队列

在传统的操作系统中,每次发生磁盘读取时,操作系统会使用某种算法确定一个“预读量”,并按照“预读量”将后续的磁盘块读入,从而在连续读取时提高系统效率.这种做法至少存在以下两个问题:

- 为了不过多地增加系统 CPU 负载,判断“预读量”的算法通常较为简单,其准确率难以保证;
- 如果预读的磁盘块在后续操作中暂时没有使用,则预读不仅占用了 IO 总线,也占用了内存缓冲区,造成浪费.

与操作系统的“预读”不同,内存网格中预取机制的基本过程是:

1) 用户节点发起一个读操作请求.
2) 内存节点接收到这一请求后,把用户节点要读的页帧发回.
3) 内存节点根据用户节点的请求,判断用户节点可能会后续读取的页帧,称为预取页帧.由于内存节点的 CPU 通常都处于空闲状态,判断预取页帧的算法可以相对比较复杂,以提高准确率.
4) 内存节点把预取页帧发送到用户节点,这是一种基于“推”数据的预取策略.
5) 用户节点收到预取页帧后,需要决定是否接收:如果接收,则使用本地物理内存保存这些预取页帧;否则,可以丢弃这些预取页帧.
6) 如果预取页帧被接收,且用户节点在后续读取中包含了这些页帧,则网络传输开销将被隐藏,性能会大为提高.

在上述过程中,用户节点需要以一定的策略来决定是否接收预取页帧,这一策略直接影响到整个预取机制的性能.如果无限制地接收,则需要耗费大量本地内存,令物理内存不足的现象更加恶化;如果丢弃的页帧过多,则预取机制又将失去效果,还存在浪费网络带宽的负面作用.为此,我们提出了“预取队列”的概念:预取队列是由一定的本地空闲内存页组成的队列,其最大长度为某个预先定义的阈值 $k$,$k$ 不超过系统当前空闲的物理内存页帧数.用户节点按照以下策略对预取队列进行更新:

1) 当用户节点收到一个预取页帧,且预取队列长度小于 $k$ 时,申请一个空闲内存页存放该预取页帧,并将其置入预取队列尾部;
2) 当用户节点收到一个预取页帧,且预取队列长度等于 $k$ 时,把预取队列头部的内存页置入预取队列尾部,并丢弃这块内存页的原有内容,以存放当前的预取页帧;
3) 当用户节点访问到了一个预取页帧时,该页帧将进入操作系统的 IO 缓存.因此,把它从预取队列中删除.

当 $k\geq 1$ 时,预取队列将接收每个预取页帧,但同时也通过队列长度的阈值 $k$ 淘汰了尚未使用的预取页帧,限制了对空闲内存页的消耗.$k$ 是影响系统策略的一个关键因子,事实上,$k$ 的取值应该与用户节点当前剩余的内存容量有关.我们在后文的模拟实验中,将对 $k$ 值和用户节点的剩余内存的比例关系进行探讨.

### 2.3 用户模式

内存节点在判断预取页帧的过程中,可以参考相应的用户节点在历史上曾经发生过的读取页帧的信息.我们把这些历史信息看作是一种用户模式,如果结合当前的读取命令,则可以采用模式匹配的方法判断出历史信息中最为接近的模式,并据此确定当前的预取页帧.我们首先考虑用户模式的组成、采集和管理,然后在下一节中讨论根据用户模式确定预取页帧的算法.

每个用户节点在使用内存节点的过程中,其读取页帧的记录都会构成一个序列,记为$\langle b_1,b_2,b_3,\ldots,b_n\rangle$,其中,$b_i(1\leq i\leq n)$是用于标识页帧的磁盘块号.与此同时,还可以记录一个相应的时间序列$\langle t_1,t_2,t_3,\ldots,t_n\rangle$,其中,$t_i(1\leq i\leq n)$表示读取页帧 $b_i$ 的时间点.如果两个相邻的时间点 $t_i$ 和 $t_{i+1}$ 有如下关系:$t_{i+1}-t_i\geq t_s$,其中,$t_s$ 是一个预先定义的阈值,则可以把序列$\langle b_1,b_2,b_3,\ldots,b_n\rangle$分为两个子序列$\langle b_1,b_2,\ldots,b_i\rangle$和$\langle b_{i+1},b_{i+2},\ldots,b_n\rangle$.这样划分的原因在于:由于读取 $b_i$ 和 $b_{i+1}$ 的时间相差较远,我们认为 $b_{i+1}$ 已经不是 $b_i$ 的后续页帧;否则,如果在用户节点读取 $b_i$ 时把 $b_{i+1}$ 当作预取节点,则 $b_{i+1}$ 可能会在预取队列中保留的时间过长,从而削弱了预取的意义.

对于 IO 密集型应用来说,虽然操作系统提供了缓存机制,但其效果仍然有限,特别是在系统空闲的物理内

存不足的情况下更为明显.这样,用户节点读取远程内存页帧的序列可能会比较长,记录这个序列需要耗费较多的存储空间.但由于大多数页帧的读取存在连续性,我们可以采用简单的行程编码方式对读取序列进行压缩,如序列⟨1,2,3,4,5,6,7,8,9,10⟩可以压缩表示为⟨1,10⟩.除此之外,Manning 等人还提出了一种实用的推理算法[26],可以采用语法分析树的形式对序列进行压缩.无论是行程编码还是推理算法都能够在线性时间内完成,这对于超长序列的处理将是非常理想的.

为了尽可能地减少用户节点的负担,记录读取序列的工作也需要由内存节点完成.当内存节点停止提供服务时,它需要把已记录的读取序列发送到用户节点,由后者进行归档,成为历史序列.随着归档的历史序列越来越多,在用户节点中将形成一个用户模式库.我们可以指定用户模式库中所包含的历史序列的最大数量,当超过这个数量时,则丢弃库中最老的序列.当用户节点每次开始使用内存节点提供的服务时,还需要把当前积累的用户模式库发送到内存节点,以便后者进行预取页帧的判断.显然,用户模式库的收集、归档和发送都需要花费额外的计算、存储和网络资源,但我们可以采取一定的技巧,隐藏这些资源的占用时间,具体过程本文不再赘述.

## 3 预取算法

预取算法是指内存节点在得到用户模式库的基础上,根据该模式库对用户节点可能会读取的页帧进行预测的过程.我们对预取算法的数学模型形式化描述如下:

设 $I=\{b_1,b_2,b_3,\ldots,b_m\}$,$I$ 是所有项的集合,$I$ 中每个项 $b_i(1\le i\le m)$的物理意义是内存网格中的页帧对应的磁盘块号;$S=\langle b_1,b_2,b_3,\ldots,b_n\rangle$,$S$ 称为一个序列,函数 $length(S)$是指 $S$ 中项的个数,称为 $S$ 的长度;对于两个序列$\alpha=\langle a_1,a_2,\ldots,a_r\rangle$和$\beta=\langle b_1,b_2,\ldots,b_s\rangle$,定义包容符号$\alpha\subseteq\beta$,其含义为存在 $r$ 个整数 $i_1,i_2,\ldots,i_r$,有 $1\le i_1<i_2<\ldots<i_r\le s$,且 $a_1=b_{i_1}$,$a_2=b_{i_2},\ldots,a_r=b_{i_r}$;定义序列连接符号$\alpha+\beta$,其结果是$\alpha+\beta=\langle a_1,a_2,\ldots,a_r,b_1,b_2,\ldots,b_s\rangle$.用户模式库是指由多个序列构成的序列,记为 $L=\langle S_1,S_2,\ldots,S_n\rangle$,其中,$S_1,S_2,\ldots,S_n$ 都是由项构成的序列.如果对于序列 $S$,存在 $S_i\in L$,且有 $S\subseteq S_i$,则称用户模式库 $L$ 支持序列 $S$;如果用户模式库 $L$ 中有多个互不相同的序列 $S_i$ 都满足 $S\subseteq S_i$,则称这些序列的个数为用户模式库 $L$ 对序列 $S$ 的支持度.

预取算法要解决的问题是:给定一个当前的页帧读取操作,假设被读取的页帧为 $b_c$,需要找到一个序列 $S_p=\langle b_c,b_1,b_2,b_3,\ldots,b_n\rangle$,满足 $length(S_p)\ge 2$,且 $S_p$ 在当前用户模式库 $L$ 中的支持度为最大.如果存在这样的 $S_p$,则称 $S_p$ 为预取序列,$b_1,b_2,b_3,\ldots,b_n$ 即为求得的预取页帧.

事实上,上述问题的背景与数据挖掘领域中的序列模式挖掘非常类似,序列模式挖掘的概念和基本算法由 Rakesh 等人于 1995 年首次提出[16],很多后续工作对基本算法进行了改进,特别是 Pei 等人提出的 PrefixSpan 算法[25],主要面向超长序列的算法时空开销进行了重点改进.但是,我们的预取问题和序列模式挖掘还存在以下两点关键的区别:

- 在序列模式挖掘中,一个序列中的元素不仅可能是一个项,还有可能是若干个项构成的集合;而在预取问题中,序列中的元素总是由单一的项组成.
- 在序列模式挖掘中,要解决的问题是给定一个阈值 $H$,找出所有支持度大于 $H$ 的序列;而在预取问题中,则是要找出满足一定条件的序列中支持度最大者.

以上两条区别在搜索空间中都是很强的剪枝条件,使得解决预取问题的算法与原始的序列模式挖掘算法相比,其时空开销更小,实用性更高.为了让预取算法更贴近实际,我们再补充以下两个条件:

- 令 $length(S)\le k+1$,其中,$k$ 为用户节点的预取队列最大长度.显然,如果一次预取的页帧数量超过了预取队列最大长度,则必然会有一部分预取页帧被立刻抛弃,这就失去了预取的意义.
- 我们已经对$\alpha\subseteq\beta$进行了定义,$\alpha\subseteq\beta$的含义为:存在 $r$ 个整数 $i_1,i_2,\ldots,i_r$,有 $1\le i_1<i_2<\ldots<i_r\le s$,令 $a_1=b_{i_1}$,$a_2=b_{i_2},\ldots,a_r=b_{i_r}$.现在,对原始的定义增加一个新的条件:对于任意 $x$ 满足 $1\le x\le r-1$,都必须有 $i_{x+1}-i_x\le d$,其中,$d$ 是一个给定的阈值.这一条件的意义是,如果相邻的两个预取页帧在用户模式库中相距较远,则两者被实际读取的时间也可能相差太远,从而使后者在预取队列中停留时间过长,失去预取的意义.

在上面这些限制条件的基础上,我们借鉴面向超长序列的 PrefixSpan 算法,提出了面向内存网格预取问题

的算法,称为 RGPSP(RAM grid prefixspan prefetching)算法.算法流程如下:

**RGPSP** 算法.

输入:用户模式库 $L$,预取队列最大长度 $k$,相邻项阈值 $d$,当前读取的页帧 $b_c$.

输出:预取序列 $S_p=\langle b_c,b_1,b_2,b_3,\ldots,b_n\rangle$,满足 $2\le length(S)\le k+1$.

算法过程:

1) 令 $S_p=\langle b_c\rangle$.
2) 令 $prefix=S_p$,$last=b_c$.
3) 令 $L'=L$.
4) 如果 $length(S)\ge k+1$,则算法结束.
5) 对于每个 $S_i\in L'$,若 $last\in S_i$,则查找变量 $last$ 在 $S_i$ 中的首次出现,删除 $S_i$ 中 $last$ 首次出现之前的所有元素和 $last$ 本身.
6) 若对于 $S_i\in L'$,$last\notin S_i$,则从 $L'$中删除 $S_i$.
7) 若 $S_p=\langle b_c\rangle$,则查找所有项 $P\in S_i$,其中,$S_i\in L'$.令序列$\langle P\rangle$在 $L'$中的支持度最大.
8) 若 $S_p\ne\langle b_c\rangle$,则查找满足如下条件的项 $P$:对于任意 $S_i\in L'$都有 $P\in S_i$.若不存在这样的项 $P$,则算法结束.
9) 置 $S_p=prefix+\langle P\rangle$.
10) 转向第 4)步.

一般的 PrefixSpan 算法需要执行递归搜索,效率较为低下.而在 RGPSP 算法中,我们只关注当前支持度最大的前缀,对于支持度不是最大的前缀,则不进行任何试探性的扩展,这就极大地提高了搜索效率.RGPSP 算法的正确性证明如下:

引理. 序列$\alpha+\beta$的支持度总是小于序列$\alpha$的支持度和序列$\beta$的支持度.

证明:对于任意序列 $S_i\in L$,如果有$\alpha+\beta\subseteq S_i$,则根据包容符号的定义,必然有$\alpha\subseteq S_i$,且$\beta\subseteq S_i$;反之,若$\alpha\subseteq S_i$,但未必有$\alpha+\beta\subseteq S_i$.故根据支持度的定义,原命题得证. □

根据引理,显然能够得到下面的定理:

定理. 如果序列$\alpha$的支持度不是最大的,则序列$\alpha+\beta$的支持度也不是最大的.

根据上述定理,如果任意前缀$\alpha$的支持度不是最大的,则由$\alpha$扩展出的序列,其支持度也不会为最大,故不必扩展以$\alpha$为前缀的序列.这说明了 RGPSP 算法的正确性.

## 4 模拟和结果

### 4.1 模拟设置

为了应用实际的系统运行状态对我们的预取机制进行模拟,首先需要实际运行一些有代表性的 IO 密集型应用,并采集它们对磁盘的访问情况.这些运行状态将构成模拟程序的重要参数.

我们使用的应用程序分别是 gcc(GNU C/C++编译器,采集它编译 Linux 内核时的磁盘访问情况),dbench (GNU 的磁盘性能测试软件之一,采用默认配置参数)和 diskWriggler(GNU 的一种磁盘吞吐量测试工具,通过在一个大文件中形成大量视频帧,并再次读取这些视频帧从而进行测试).为了记录这些程序对磁盘的访问,我们修改了 Linux 2.4.22 版本的内核,并记录了每次对文件系统的读操作(即使一个读操作命中了文件系统的缓存,也同样需要记录下来),记录下的信息包括每次读操作的磁盘块号和发出该操作的系统时间.记录的过程中使用了一台具有 Intel Pentium 3.0GHz CPU,1GB 内存的 PC 机,使用的操作系统是 RetHat 9.0(操作系统内核已更换为修改后的 2.4.22 版本).

在我们之前的工作中,已经采用离散事件模拟的方式模拟出了具有 1 000 个节点的内存网格环境.其中包括 PC、工作站和集群 3 种不同类型的节点,三者的比例为 89:10:1,一个集群还包括 32 个子节点.每个节点或子节点的物理内存根据节点类型随机生成,从 128M 到 4G 不等.网络拓扑是采用 BRITE 软件的 ASWaxman 模式生成的一个含边权重的无向图,如果两个节点在图中没有直接相连,则采用 Dijkstra 算法模拟一个最短路由.网

络延迟和带宽的平均值取校园网上实际测试的典型参数,后文将详细加以描述.对于这样一个模拟系统,如果再具体考虑每个用户节点的预取操作,则模拟程序的资源消耗会非常严重.而由于当每个用户节点分别使用远程内存时,其使用过程是相互独立的,不失一般性,我们可以在每个时刻只关注一个随机选取的用户节点和相应的内存节点,集中考虑其工作过程中网络传输效率的问题.待这个用户节点和内存节点之间的交互结束后,再重新考虑另外一对节点.

在 2.4.x 内核版本的 Linux 操作系统中,提供了对文件系统的缓存机制,缓存的最大容量为系统剩余的内存容量.在我们的模拟过程中,也对操作系统提供的缓存机制进行了模拟,系统缓存的替换策略采用 LRU(least recently used)方法.

模拟过程中一些重要参数的取值主要参照了我们之前的工作,如下所述:

对于一次磁盘读取操作,假如要读的是 $n$ 个连续的磁盘块,那么需要耗费的时间 $O_d$ 用下式进行计算:

$$O_d = T_S + T_L + (n-1)\times T_W + n\times\frac{S_p}{B_d}.$$

在上式中,$T_S$ 是指磁盘的平均寻道时间,也就是磁头移动到相应的磁道上需要的时间;$T_L$ 是指平均等待延迟,相当于磁盘旋转半周的时间;$T_W$是读取两个相邻的磁盘块时需要的等待时间;$S_p$是一个磁盘块的大小;$B_d$是磁盘外部传输速率,受到磁盘接口和总线技术的影响.对于一个典型的 10K 转速的商用硬盘,其典型参数为 $T_S$=4.9ms,$T_L$=3.0ms,$T_W$=0.2ms,$S_p$=4KB,$B_d$=80MB/s.

对于网络传输来说,传输 $n$ 个磁盘块需要耗费的时间 $O_n$ 用下式进行计算:

$$O_n = T_U + T_{RTT} + n\times\frac{S_P}{B_N}.$$

在上式中,$T_U$是启动网络传输所需要的时间;$T_{RTT}$是网络传输的双向延迟时间,也就是说,相当于平均网络延迟的 2 倍;$B_N$ 是网络带宽.我们在内存网格中考虑的广域网通常是指企业内部或校园内部的高速互联网络,与目前的 Internet 相比,具有带宽较高、延迟较低的特点.根据在校园网上的实际测试结果,取 $T_U$=5μs,$T_{RTT}$=3.0ms,$B_N$=4.0MB/s.考虑到网络性能的不稳定性,我们在模拟过程中对上述参数进行了扰动.

### 4.2 结果及分析

我们首先对内存网格在密集 IO 应用中的缓存作用进行评价.由于系统缓存通常利用本地的剩余内存建立,可以想象:当本地内存趋于不足时,系统缓存的作用也相应降低.我们通过内存网格建立二级缓存,并分别对启用和不启用 RGPSP 预取算法的情况进行模拟.

在图 3 中,分别用不同灰度的柱形代表了仅使用传统的缓存机制而不使用内存网格、使用内存网格作为二级缓存但不启用预取、使用内存网格作为二级缓存且启用预取算法的情况.在标准配置下,我们设定系统的本地剩余内存为 400MB,同时,对剩余内存减少或增加的情况进行了模拟.由图 3 可见,当不使用内存网格时,IO 的缓存作用受系统的剩余内存容量影响非常大.当系统的剩余内存为 80MB 时,操作系统的缓存趋于失效,IO 的时间开销是剩余内存为 560M 时的 3 倍以上;当使用内存网格后,由于引入了二级缓存,使得 IO 开销受剩余内存的影响有所减缓;如果开启内存网格的预取机制,由于内存网格的传输开销被有效地隐藏起来,即使在本地剩余内存严重不足的情况下,IO 开销的增长也非常有限.

当本地剩余内存大于 640MB 时,三者之间几乎没有差别.这是因为本地内存构成的磁盘缓存已经能达到较好的效果,内存网格构成的二级缓存基本上没有起到作用.

在我们的预取算法中,预取队列是一个非常关键的因素.考虑到预取队列的最大长度应该与本地剩余内存呈比例关系,我们分别考虑预取队列从最大占用剩余内存的 1/2 到最大占用 1/64 的不同情况(标准配置下该比例取 1/5).结果如图 4 所示,两条曲线分别代表了在不同配置下,系统的 IO 开销和两级 IO 缓存共同的命中率.

可以看出:当预取队列最大长度占剩余内存的 1/8 以上时,系统 IO 开销变化不大;而当该比例低于 1/8 时,系统 IO 开销迅速增长,命中率也随之下降.这是因为预取队列的最大长度代表了预取页帧在该队列中的停留时间,当预取队列长度超过剩余内存的 1/8 时,大多数预取页帧已经能够停留足够多的时间,使得从队列中淘汰前

能够被用户节点实际读取;而当低于 1/8 时,虽然节省了一部分本地内存,但页帧在预取队列中淘汰过快,使得预取算法不能很好地发挥作用.

用户模式库的大小决定了用户节点能够存储的历史序列的数目,并直接影响到 RGPSP 预取算法的计算开销和精度.我们把这一参数在 1 000~8 000 之间进行变化(标准情况下该参数取 3 000),结果如图 5 所示.

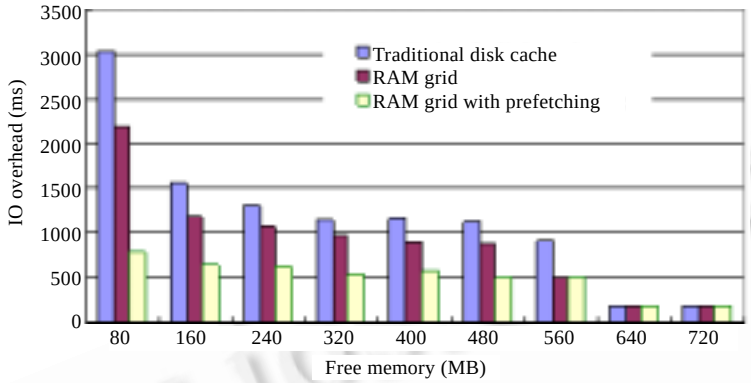


Fig.3 The relationship of the caching effect of RAM grid and local free memory

图 3 内存网格的缓存作用与本地剩余内存的关系

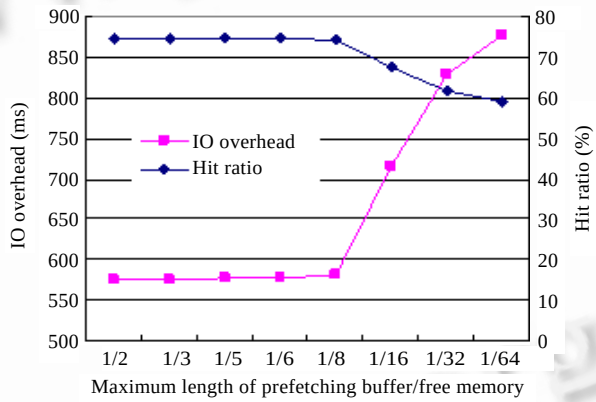


Fig.4 The effect of maximum prefetching buffer length for prefetching algorithm of RAM Grid

图 4 内存网格的预取算法受预取队列最大长度的影响

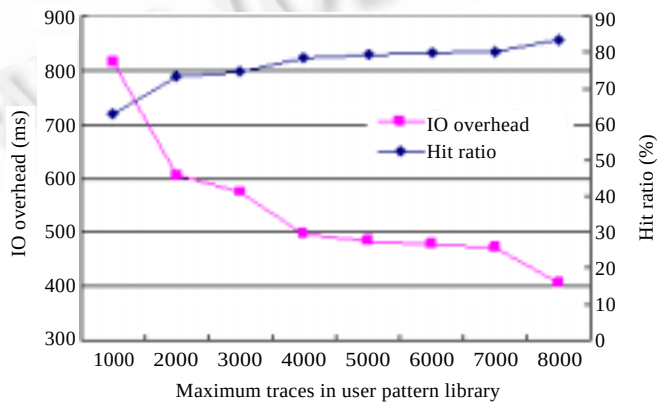


Fig.5 The effect of maximum traces in user pattern library for prefetching algorithm of RAM Grid

图 5 内存网格的预取算法受用户模式库最大数量的影响

显然,用户模式库中的历史序列数目越多,预取的准确率也越高.但从图 5 中可以看出:当用户模式库的最大数量超过 4 000 后,IO 开销和缓存的命中率都变得较为平缓;而更大的用户模式库导致传输和计算的开销都变得更高,从而带来更多潜在的负面作用.应根据实际情况进行权衡,选择合适的用户模式库大小.

## 5 结束语

内存网格是传统网格计算技术的扩展.它利用广域网上高速互联的大量空闲内存节点,解决内存密集型和 IO 密集型应用的性能受物理内存容量影响的问题.内存网格的本质是使用远程内存填补本地内存和磁盘之间的性能差异,其应用效果主要受网络通信开销的影响.

预取技术是有效隐藏网络通信开销的手段之一,它广泛应用于计算机系统的各个层次.我们在内存网格中引入“推”数据的方法和用户模式的分析,实现高效的数据预取机制;并通过分析内存网格中的一些细节问题,提出了一种基于序列模式挖掘的预取算法,证明了其正确性.本文还通过基于实际运行状态的模拟,对该算法的重要参数和效果进行了探讨和验证,结果表明,在内存网格中引入合适的预取技术,对内存网格的实际应用效果具有重要的价值.

**References**:


[1] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. Int’l Journal of High Performance Computing Applications, 2001,15(3):200−222.

[2] Frey J, Tannenbaum T, Livny M, Foster I, Tuecke S. Condor-G: A computation management agent for multi-institutional grids. In: Proc. of the 10th IEEE Int’l Symp. on High Performance Distributed Computing. San Francisco: IEEE Computer Society, 2001. http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/hpdc/2001/1296/00/1296toc.xml&DOI=10.1109/HPDC.2001.945176

[3] Baru C, Moore R, Rajasekar A, Wan M. The SDSC storage resource broker. In: Proc. of the CASCON’98 Conf. IBM Press, 1998. http://citeseer.ist.psu.edu/64825.html

[4] Foster I, Kesselman C. The Grid 2: Blueprint for a New Computing Infrastructure. 2nd ed., Morgan Kaufmann Publishers Inc., 2003.

[5] Chu R, Xiao N, Zhuang Y, Liu Y, Lu X. A distributed paging RAM grid system for wide-area memory sharing. In: Proc. of the 20th Int’l Parallel and Distributed Processing Symp. Toronto: X-CD Technologies Inc., 2006.

[6] Feeley MJ, Morgan WE, Pighin FH, Karlin AR, Levy HM, Thekkath CA. Implementing global memory management in a workstation cluster. In: Proc. of the Symp. on Operating Systems Principles. New York: ACM Press, 1995. 201−212.

[7] Markatos EP, Dramitinos G. Implementation of a reliable remote memory pager. In: Proc. of the USENIX Annual Technical Conf. 1996. 177−190. http://www.usenix.org/publications/library/proceedings/sd96/

[8] Acharya A, Setia S, The utility of exploiting idle memory for data-intensive computations. Technical Report, TRCS98-02, Santa Barbara: University of California at Santa Barbara, 1998.

[9] Foster I. Service-Oriented science. Science, 2005,308(5723):814−817.

[10] Patterson DA. Latency lags bandwith. Communications of the ACM, 2004,47(10):71−75.

[11] Chilimbi TM, Hirzel M. Dynamic hot data stream prefetching for general-purpose programs. In: Proc. of the ACM SIGPLAN 2002 Conf. on Programming Language Design and Implementation. New York: ACM Press, 2002. 199−209.

[12] Wenisch TF, Somogyi S, Hardavellas N, Kim J, Ailamaki A, Falsafi B. Temporal streaming of shared memory. In: Proc. of the 32nd Annual Int’l Symp. on Computer Architecture. Los Alamitos: IEEE Computer Society, 2005. 222−233.

[13] Shi X, Yang Z, Peir JK, Peng L, Chen YK, Lee V, Liang B. Coterminous locality and coterminous group data prefetching on chip-multiprocessors. In: Proc. of the 20th Int’l Parallel and Distributed Processing Symp. Toronto: X-CD Technologies Inc., 2006.

[14] Swaminathan N, Raghavan SV. Intelligent prefetch in WWW using client behavior characterization. In: Proc. of the 8th Int’l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Los Alamitos: IEEE Computer Society, 2000.

[15] Yu SZ, Kobayashi H. A new prefetch cache scheme. In: Proc. of the IEEE Global Telecommunications Conf. 2002. 350−355. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=892028

[16] Agrawal R, Srikant R. Mining sequential patterns. In: Proc. of the 11th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Computer Society. 1995.

[17] Dahlin MD, Wang RY, Anderson TE, Patterson DA. Cooperative caching: Using remote client memory to improve file system performance. In: Proc. of the 1st Symp. on Operating Systems Design and Implementation. 1994. 267−280. http://citeseer.ist.psu.edu/26547.html

[18] Anderson TE, Dahlin MD, Neefe JM, Patterson DA, Roselli DS, Wang RY. Serverless network file systems. ACM SIGOPS Operating Systems Review, 1995,29(5):109−126.

[19] Flouris MD, Markatos EP. The network RamDisk: Using remote memory on heterogeneous NOWs. Cluster Computing, 1999,2(4): 281−293.

[20] Oleszkiewicz J, Xiao L, Liu Y. Parallel network RAM: Effectively utilizing global cluster memory for large data-intensive parallel programs. In: Proc. of the 2004 Int'l Conf. on Parallel Processing. Los Alamitos: IEEE Computer Society, 2004. 353−360.

[21] Voelker GM, Anderson EJ, Kimbrel T, Feeley MJ, Chase JS, Karlin AR, Levy HM. Implementing cooperative prefetching and caching in a globally-managed memory system. In: Proc. of the Joint Int'l Conf. on Measurement and Modeling of Computer Systems. New York: ACM Press, 1998. 33−43.

[22] Yang CL, Lebeck AR, Tseng HW, Lee CH. Tolerating memory latency through push prefetching for pointer-intensive applications. ACM Trans. on Architecture and Code Optimization, 2004,1(4):445−475.

[23] Agrawal R, Srikant R. Mining sequential patterns: Generalizations and performance improvements. In: Apers PMG, Bouzeghoub M, Gardarin G. Proc. of the 5th Int'l Conf. on Extending Database Technology: Advances in Database Technology. London: Springer-Verlag, 1996. 3−17.

[24] Ayres J, Flannick J, Gehrke J, Yiu T. Sequential pattern mining using a bitmap representation. In: Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2002. 429−435.

[25] Pei J, Han J, Mortazavi-Asl B, Pinto H. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. of the IEEE 17th Int'l Conf. on Data Engineering. 2001. 215−226.

[26] Nevill-Manning CG, Witten IH. Linear-Time, incremental hierarchy inference for compression. In: Proc. of the Conf. on Data Compression. Los Alamitos: IEEE Computer Society, 1997.

褚瑞(1979－),男,陕西西安人,博士生,主要研究领域为网格计算,高性能并行分布处理技术.

卢锡城(1946－),男,教授,博士生导师,中国工程院院士,CCF 高级会员,主要研究领域为 MPP 实现技术,分布处理技术,网络技术.

肖侬(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网格计算和数据网格,高性能并行分布处理技术.