

高性能计算中一种改进的数据访问节能技术研究

邓定胜

(四川民族学院计算机科学系 康定 626001)

摘 要 针对现有磁盘功率管理方法无法处理高能耗并行应用的短空闲周期、大规模的代码更改和能耗开销较大等问题,提出了一种面向编译器的数据访问调度技术。该技术包括两个阶段:在第一阶段,编译器分析并行应用程序,提取磁盘访问模式,然后生成调度表;在第二阶段,“数据访问调度器”根据调度表进行数据访问。与先前基于软件的策略相比,所提方法不需要改变代码或数据结构。实验评估结果表明,对于数据密集型工作负载,所提方法可以有效提升节能效果,节能率从 5.5% 上升到 11.8%,从而增加了磁盘降速策略对数据密集型高性能计算的可行性。此外,它也将多速磁盘的节能效果从 12.7% 增加到了 27.6%。

关键词 磁盘功率管理,空闲周期,代码,数据访问,编译器,能量

中图分类号 TP392 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.2.041

Research on Improved Energy-saving Technology for Data Access in High Performance Computing

DENG Ding-sheng

(Department of Computer Science, Sichuan University for Nationalities, Kangding 626001, China)

Abstract Aiming at the problems that the existing disk power management methods cannot handle short idle periods of high-performance parallel applications, require extensive code modifications and cost much energy, this paper proposed a compiler-directed data access scheduling technology for saving disk energy. The proposed technology contains two phases. In the first phase, the compiler analyzes parallel application programs, extracts disk access patterns and generates scheduling tables, while in the second phase, a “data access scheduler” performs the actual data accesses according to scheduling tables. As compared to prior software based efforts, our framework requires no code or data restructuring. Our experimental evaluation reveals that the proposed framework effectively increases the energy savings brought by the disk power down mechanism for data-intensive workloads, and the rate of energy savings improves from 5.5% to 11.8%, making disk spin-down a viable strategy in data-intensive high-performance computing. In addition, it increases the energy benefits of multi-speed disks from 12.7% to 27.6%.

Keywords Disk power management, Idle periods, Code, Data access, Compiler, Energy

鉴于冷却和成本因素,能耗已经成为大规模高性能分布式计算环境需要考虑的重要方面^[1]。先前大部分研究都针对处理器组件^[2]、片上网络和片对片网络^[3]及内存组件而展开,磁盘功耗获得的关注相对较少。然而,磁盘功率是功率总预算的重要部分,对数据密集型计算服务器而言更是如此,因为该种服务器经常需要存储、读取和处理位于磁盘中的数据。文献[4]的研究结果表明,存储数据时消耗的能量占数据中心总能耗的 1/3 以上,对数据中心服务器上的存储密集性应用来说,这一比例可能达 70%。因此,对基于磁盘的存储系统的能耗问题展开研究具有重要意义。

当前关于磁盘功率管理的大部分研究,重点放在硬盘空闲周期的使用上^[5]。主要思路是:如果检测出的空闲时间超过设定阈值,则使磁盘进入睡眠模式或低功率模式。基于这一思路的一种策略是:当到达预设空闲阈值时,使磁盘完全降速。还有另一种策略是使用多速磁盘^[6],并在运行期间改变磁盘转速。多速磁盘相对磁盘降速的优势是,它可以使用更

短的空闲周期。因为一般来说,降低磁盘速度所需的成本,要比使磁盘完全降速再提速的成本低。为了提升这些硬件功率节约机制的效能,可以改变代码和数据的布局,以延长磁盘空闲周期的长度,文献[7,8]证明了这一策略在部分应用中取得了显著效果。然而,对于并行程序来说,更改代码的数据结构必须要考虑相关性、数据痕迹和同步等因素,因此全面更改程序代码的工作量很大。为此,本文提出并评估了一种编译器驱动的应用层数据访问(I/O 调用)调度技术,以节约磁盘能量。

1 相关工作

节能数据访问技术是目前的研究热点,相继有众多学者提出了一系列有代表性的方案。如文献[9]提出了适用于顺序数据访问的节能磁盘阵列 S-RAID5,它采用局部并行策略:阵列中的存储区被分成若干组,组内采用并行访问模式。在 S-RAID5 磁盘阵列中运行磁盘调度算法,辅以合适的 Cache

到稿日期:2014-04-03 返修日期:2014-07-03 本文受四川省教育厅自然科学一般项目:量子程序中一种改进的延时估计算法研究(15ZB0332),四川省教育厅自然科学重点项目:基于粗糙集理论的数据挖掘算法研究(13ZA0136)资助。

邓定胜(1978—),男,副教授,主要研究方向为数据库应用技术、大数据处理、算法分析与程序设计、计算机网络、数字图像处理等。

策略来过滤少量的随机访问。文献[10]提出一种由 SSD 固态硬盘与普通磁盘组成的混合 S-RAID 结构,通过关闭部分处于空闲状态的磁盘,达到节能效果。文献[11]提出了一种基于预读策略的节能数据访问技术。引入文件系统数据访问中的预读方式,读取数据并将其聚合到一起进行访问,减少设备组件的状态切换,从而实现降低能耗的目的。文献[12]提出了一种基于布局的虚拟磁盘节能调度方法。该方法将磁盘阵列动态划分为工作区与就绪区,以工作区为主向用户分发资源,并以未连接虚拟机的虚拟磁盘为单位,根据实时负载情况对虚拟磁盘布局进行动态优化。实验结果表明,这种方法不仅能够降低磁盘阵列的能耗,而且能够有效地缓解响应时间延长的问題。Weissel 等人在文献[13]中提出协作式 I/O 策略,该策略指明哪些 I/O 操作可延迟或可取消,以便底层的磁盘功耗管理机制可以利用这一信息来降低能耗。文献[14]提出使用预取和缓存策略来提高 I/O 访问模式的猝发性,以节约能量。Pinheiro 等人在文献[15]中提出一种称为常用数据集中存储(PDC)的节能技术,即将访问最为频繁的磁盘数据动态转移到部分磁盘上。Son 等人提出多种基于编译器的代码变换和预取技术,以节约磁盘能耗^[16]。本文在现有研究工作的基础上,提出了一种改进的数据访问调度方法,该方法不需改变任何代码或数据布局,在运行期间选择性地发出 I/O 请求,同时在编译期间调度数据访问。最后通过仿真实验验证了该方法的有效性。

2 磁盘节能机制

本文的研究架构如图 1 所示,应用程序对多个客户端节点实现并行化(每个客户端节点一个进程),底层并行文件系统在 I/O 节点间使用数据条策略(即每个文件被分为多个块,称为“数据条”,这些数据条按循环规则分布于 I/O 节点间)。虽然 I/O 节点为了提升性能和可靠性在磁盘内将数据块分为更多的数据条,但是我们只关注 I/O 节点层面的数据访问模式和磁盘功率管理,因为 I/O 节点层面的数据条策略对编译器是开放的,通过 I/O 中间件的 API 函数可以对数据条进行控制。

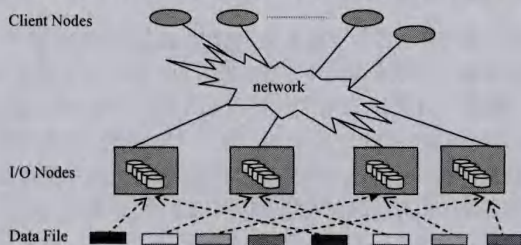


图 1 I/O 存储架构和数据文件分块

节约磁盘能量的传统方法是在一个空闲周期后对磁盘降速,因为磁盘的主轴电机在运行期间会消耗大量能量^[5]。这一策略的缺点之一就是:经过降速的磁盘在服务下一个 I/O 请求前,必须再次提速,而这一步骤会消耗大量时间。之前对服务器的实验也表明,因为数据密集型应用空闲周期较短,所以这一策略对高性能计算的效果非常有限^[6]。另一种方法是使用多速策略^[17],以利用短空闲周期,通过动态调整旋转周期,使得即使在低转速条件下也可以满足请求。虽然文献[6]等证明了对于数据密集型工作负载,多速策略优于降速策略,但是如果磁盘的空闲周期更长,则两种策略的节能效果将会进一步提升,性能会更为优异。因此,本文研究的主要目的就

是在应用层通过对磁盘进行调度,来延长磁盘空闲周期的长度。

3 本文方法概述

图 2 给出了本文方案的框架。该方法主要有两个部分:优化编译器和运行期间数据访问调度器。编译器在代码和 I/O 并行化后进行磁盘功率优化,这一过程分为两步:确定访问松弛度和数据访问调度。第一步确定访问松弛度或访问区域(用循环迭代数量表示)。该区域表示磁盘数据的数据源和数据用户间的时间间隔,也就是“上次数据写入”到“数据读取”间的时间间隔。如果区域超过一个迭代周期,则可以对迭代进行选择以调度该次数据访问。很明显,区域越大,数据访问调度的灵活性越大。在确定了所有数据访问的松弛度后,第二步就是确定每次数据访问的具体时间点,并为每个应用进程存储该信息于表中。在决定这些时间点时,我们的目标是延长磁盘空闲周期的长度。

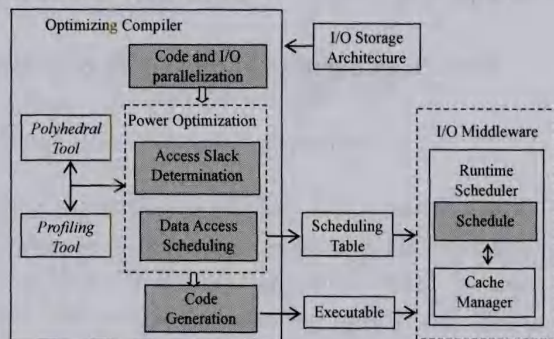


图 2 本文方法概览

部署于 MPI-IO 库^[18]之上的数据访问调度器在运行期间根据生成的调度表来访问数据,并为应用进程缓存数据。为了降低缓存开销,对数据访问来说,只有在调度时被安排的迭代时间早于它在程序中的初始时间点,该次数据访问才会被调度器执行。更具体地说,该调度器是程序进程开始运行时创建于各个客户端节点上的轻量级线程。它的任务是存取数据及程序进程和其他调度器线程间的通信和同步。“提前获得”的数据存储于受客户端所有调度器进程共同管理的全局缓存中,全局缓存的部署基于 Liao 等人开发的 MPI-IO 缓存库^[19]。每当应用进程发出 I/O 请求时,首先检测缓存。如果缓存命中,则立刻把数据返回给应用进程,同时设定入口失效,以便为调度器进程提前取得的后续数据腾出空间。如果缓存满,则调度器进程停止预取数据。请注意,不同客户端节点的应用进程在运行时并不采取锁步策略。因此,如果客户端节点 A 的调度器线程想要预取客户端节点 B 的线程写入数据,则它需要在从磁盘预取数据之前,首先检查节点 B 调度器进程维护的“本地时间”。这一策略可以保证运行期间调度器提供的数据是正确且有用的。由于篇幅有限,我们省略运行期间调度器的部署内容(比如缓存策略、缓存数据和元数据的管理),在下文中把重点放在优化编译器上。

4 面向数据访问调度的优化编译器

4.1 确定访问松弛度

我们的目标应用程序是处理大量磁盘数据的数据密集型高性能并行应用。这些程序的结构往往是运行于多维数组上的一系列循环(见图 3 矩阵乘法)。因此,我们使用“循环迭

代”作为数据访问调度的粒度(单元)。在不造成分歧的前提下,“迭代”、“调度点”和“调度间隙”可以互换使用。

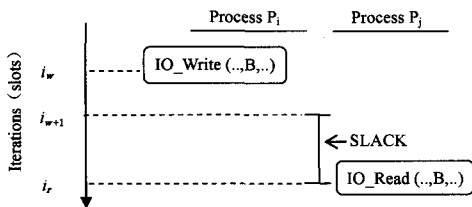
```

MPI_File_open(....., U, &fh_U, .....); // Open files, U, V, and W
MPI_File_open(....., V, &fh_V, .....);
MPI_File_open(....., W, &fh_W, .....);
for m=1, R, 1{ // Loop on horizontal file block
    MPI_File_read(fh_U, .....); // Read next block of matrix U
    for n=1, R, 1{ // Loop on vertical file block
        MPI_File_read(fh_V, .....); // Read next block of matrix V
        for i=1, N, 1 // Actual matrix product
            for j=1, N, 1
                for k=1, N, 1
                    W[i,j] += U[i,k] * V[k,j];
        MPI_File_write(fh_W, .....); // Write block of W
    }
}
MPI_File_close(&fh_U); // Close all files
MPI_File_close(&fh_V);
MPI_File_close(&fh_W);

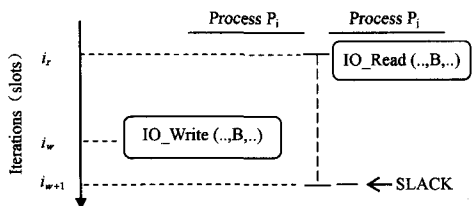
```

图3 基于MPI-IO写入矩阵乘法代码

在该例中,每个文件被分为 $R \times R$ 个数据块,每个数据块有 $N \times N$ 个元素访问磁盘数据时(即 I/O 调用),我们定义其松弛度为数据被写入磁盘的时间点和数据被切实需要的时间点(比如被读取操作需要)之间的间隔,如图 4(a)所示。在时间点 i_w ,进程 P_i 将数据块 B 写入磁盘;在时间点 i_r ,进程 P_j 从磁盘读取数据块 B 。区域 $[i_w + 1, i_r]$ 称为该读取访问的松弛度。两个进程 P_i 和 P_j 可能相同也可能不同。在前一种情况下,我们将松弛度称为进程内松弛度;在后一种情况下,我们将其称为进程间松弛度。如果读取和写入操作来自不同的进程,则由于循环并行化和迭代空间的规一化,可能在对应的写入操作前,对某一数据块的读取操作可能已经发出。因此,松弛度可能取负值,如图 4(b)所示。此时,读取操作需要等待写入操作完成,因此,只能在时间点 $i_w + 1$ 发出读取操作。于是,负值松弛度的长度为 1。很显然,我们对正值松弛度感兴趣(即松弛度的长度超过一个循环),因为只有正值松弛度可以让我们在 $i_w + 1$ 和 i_r 间(含)的任意时间点对访问点进行调度。



(a) 正值松弛度示例



(b) 负值松弛度示例

图4 松弛度

在本文方法中,我们使用 Omega 库^[20]或描述工具(profiling tool)来标识松弛度。当循环边界和数据引用是加入(encode)循环指数的仿射函数和循环独立参数时,使用多面体工具;当循环嵌套非仿射或有象征性界限,使用描述工具。如果循环非常大,为了降低调度器线程和应用进程间的同步开销及下文给出的调度算法的运行时间,我们考虑 $d(d > 1)$ 次迭代而不是一次迭代作为一个单元来衡量松弛度。

4.2 数据访问调度

当获得了程序的所有松弛度后,本文数据访问调度步骤将在每次数据访问的松弛度内确定一个调度点。此时,需要考虑各次数据访问 I/O 节点的水平重用和垂直重用。水平重用是指来自不同进程但被在同一时间点调度的数据访问从同一组 I/O 节点读取数据;而垂直重用是指调度在连续迭代循环内的数据访问从同一组 I/O 节点读取数据。请注意,垂直调用既可能发生于来自同一进程的数据访问请求,也可能来自不同进程的数据访问请求。

为了对两次数据访问间的 I/O 节点重用情况进行量化,我们为每次访问定义一个签名,并计算两次访问的签名间的距离。尤其地,目标存储架构的所有 n 个 I/O 节点依次排序,第一个为 0,最后一个为 $(n-1)$ 。每个 I/O 节点关联签名中的一个比特位。假设 $\mathcal{D}$ 表示一次数据访问将要涉及到的一组 I/O 节点(根据数据条尺寸计算出来),该次访问的签名 g 定义为: $g = [\eta_1 \eta_2 \dots \eta_{n-1}]$, 其中:

$$\eta_i = \begin{cases} 1, & \text{if } i \in \mathcal{D} \\ 0, & \text{otherwise} \end{cases}$$

如果该次数据访问用到 I/O 节点 i ,则签名 g 的第 i 位为 1,否则为 0。很显然,两次数据访问的签名告诉我们它们大量的相似性和差异性信息。尤其地,如果两个签名相同,则对应的数据访问使用完全相同的一组 I/O 节点。如果两个签名间有 n 个比特不同,则两次数据访问使用完全不同的 I/O 节点。两个签名 g_1 和 g_2 的距离定义为:

$$\text{distance}(g_1, g_2) = n - \text{similarity}(g_1, g_2) + \text{difference}(g_1, g_2)$$

其中, n 是 I/O 节点数量,相似度表示两个签名对应位置的 1 的数量,差异性表示不同比特位的数量。相似性表示的是将会重用的活跃 I/O 节点的数量,差异性表示的是将要额外开启的 I/O 节点数量。因此,距离指标同时描述了两个约束。通过该指标,可以实现活跃磁盘重用率最大化,同时实现后续迭代循环非活跃磁盘使用率最小化。

在后续小节中,我们首先假设所有数据访问(I/O 调用)的长度为 1 个循环,即它们在一个调度时隙内结束,然后给出一种基本算法,进而将算法扩展到不同长度的数据访问情况。此后,我们进一步给出一种经过改进的算法,该算法不是单纯地实现磁盘功率节约效果最大化,而是在功率和性能指标间实现平衡。

4.2.1 基本算法

已知来自不同进程的一组数据访问及其松弛度,我们的基本算法可以根据松弛度的非降序次序,从最短松弛度开始,依次确定这些数据访问的调度点。选择这样的处理次序是因为相比于长松弛度的数据访问,松弛度较短的数据访问受到的约束更多。因此考虑到 I/O 节点重用性,应该首先调度它们。相比较而言,大部分情况下,长松弛度的数据访问即使推迟处理也不会有严重的负面效果,毕竟它们的自由度更大。

对一次具体的数据访问,为了确定理想的访问点,我们检查松弛度内的所有时隙。如果某个时隙调度给同一进程的另一次数据访问,则认为该时隙不可用,然后略过。也就是说,我们避免同一时隙内调度来自同一进程的多次数据访问。然而,来自不同进程的数据访问可以调度在同一时隙内。如果时隙可用,则我们计算它的重用因子(下文将作介绍)。检查完所有的时隙后,我们选择重用因子最大的时隙作为该次数据访问的调度点。如果有多个时隙的重用因子相同,则从中随机选择一个。

数据访问松弛度内时隙 t 的重用因子 $\mathcal{R}_t$ 定义如下:

$$\mathcal{R}_t = \sum_k \left(\frac{1}{d_{t+k}} \times \sigma_{|k|} \right), k \in [-\delta, \delta] \quad (1)$$

$$d_{t+k} = \text{distance}(g, G_{t+k}), \sigma_{|k|} = 1 - \frac{|k|}{\delta + 1} \quad (2)$$

其中, $\sigma_{|k|}$ 表示权重, d_{t+k} 表示该次访问的签名 g 和群组活跃签名 G_{t+k} 间的距离, 计算为:

$$G_{t+k} = g_{(t+k)_1} | g_{(t+k)_2} | g_{(t+k)_3} | \dots | g_{(t+k)_m}$$

其中, ‘|’ 表示逻辑逐位 OR 操作, $g_{(t+k)}$ 表示已经调度到迭代 $t+k$ 处的数据访问的签名。很显然, 调度到迭代 $t+k$ 处的数据访问数量(m)不得大于进程数量, 因为在任意指定时刻只能有一个进程的一次数据访问被调度。

请注意, 上文定义的重用因子 $\mathcal{R}_t$ 包括我们之前讨论的垂直重用和水平重用。为了计算数据访问时隙 t 的重用因子 $\mathcal{R}_t$, 我们考虑了该次访问和调度在迭代时刻 $t-\delta$ 和 $t+\delta$ 间的所有其他数据调用间的重用情况, 其中 δ 是预设值(可配置), 表示我们考虑的垂直重用范围。根据相对时隙 t 的位置来确定该范围内某次迭代的权重 δ 。迭代与时隙 t 的距离越近, 则权重越大。虽然有多种权重分配方法, 但是我们根据等式(2)来确定权重。例如, 如果 $\delta=4$, 我们有 $\sigma_0=1, \sigma_1=0.8, \sigma_2=0.6$ 等等, 这就是说, 迭代 $t, \{(t-1) \text{ 和 } (t+1)\}$, 和 $\{(t-2) \text{ 和 } (t+2)\}$ 的权重分别为 1, 0.8 和 0.6, 如图 5 所示。请注意, 本文根据 I/O 节点的特征(比如空闲磁盘进入低功耗模式所需时间)来选择 δ 的值。原因是, 被 $t-\delta$ 和 t 间的迭代所用过的磁盘在迭代 t 时可能仍然处于活跃状态; 类似地, 在迭代 t 时被激活的磁盘当迭代 $t+\delta$ 结束时可能一直处于被用状态。此外, 对 $t-\delta$ 和 $t+\delta$ 间的每次迭代, 我们通过评估正被调度的数据访问的签名和群组活跃签名间的距离 d_{t+k} 来研究水平重用情况。然而, 因为距离越短, I/O 节点的重用情况越好, 相似度越高, 差异性越小, 所以我们在式(1)中使用 $\frac{1}{d_{t+k}}$ 而不

是 d_{t+k} 。请注意, d_{t+k} 可能为 0, 此时 $\frac{1}{d_{t+k}}$ 设为 2。

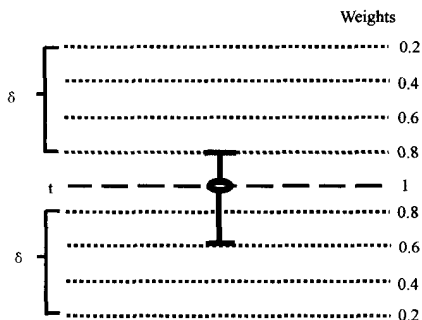


图 5 垂直重用范围和权重分配

本文基本算法的伪代码如下所示。

算法 1 数据访问调度算法

输入: IO 访问 $\mathcal{A}$, 对每次访问 a 有起始点 $a.b$, 终点 $a.e$, 签名 $a.g$, 线程

ID $a.g$, 重用范围 δ , 及该范围内的迭代权重 σ_k , 迭代总次数 N_t ;

输出: $\mathcal{A}$ 中每次访问 a 的调度点 $a.p$ 。

```

1. for  $i \leftarrow 1$  to  $N_t$  do
2.    $G_i = 0$ ; // 初始化群组活跃签名
3. end for
4. Sort  $\mathcal{A}$ ; // 按照非降序次序 ( $a.e - a.b + 1$ )
5. for  $a \in \mathcal{A}$  do
6.    $BR \leftarrow 0$ ; // 最优重用因子
7.   for  $i \leftarrow a.b$  to  $a.e$  do
8.     if  $\exists b \in \mathcal{A} \ \& \ b.p = i \ \& \ b.id = a.id$  then
9.       continue; // 该时隙不可用
10.    end if
11.     $R = \sum \frac{1}{\text{distance}(a.g, G_{t+k})} \times \sigma_{|k|}, k \in [-\delta, \delta]$ ;
12.    if  $R \leq BR$  then
13.      continue;
14.    else
15.       $BR \leftarrow R$ ;  $a.p \leftarrow i$ ;
16.    end if
17.  end for
18.   $G_{a.p} \leftarrow G_{a.p} | a.g$ ; // 更新群组活跃签名
19. end for

```

我们现在假设 3 个不同进程产生 10 次数据访问来阐述该算法如何工作。它们基于 16-I/O 的节点存储架构的松弛度和签名如图 6 和图 7 所示。

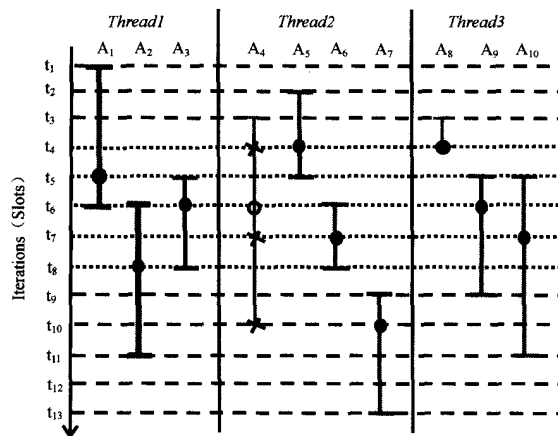


图 6 长度为 1 时数据访问调度示例

Access	Signature
A1	0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
A2	0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
A3	0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
A4	0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
A5	0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
A6	0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0
A7	1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
A8	0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
A9	0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
A10	0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0

图 7 图 6 中数据访问的签名

本文算法从 A_8 开始, 然后根据松弛长度, 处理 A_6, A_3, A_5 等等。假设除了 A_1 外的所有数据访问的高度点均被确定(图 6 中的实心圆)。现在, 我们的任务是为其确定一个理想的调度点。根据上文讨论内容, 时隙 t_1, t_7 和 t_{10} 将不被考虑

(图中的交叉处),因为已经有来自同一进程的数据访问(A_5, A_6, A_7)被调度到这些时隙。因此,我们只计算位于 A_4 松弛度内其余时隙的重用因子,并且选择数值最大的一个时隙。以 t_6 为例来说明如何计算其重用因子。假设 $\delta=2$,我们有 $\sigma_0=1, \sigma_1=1-1/(\delta+1)\approx 0.7, \sigma_2=1-2/(\delta+1)\approx 0.4$,且:

$$\begin{aligned} \mathcal{R}_6 &= \frac{1}{d_5} * 1 + \frac{1}{d_5} * 0.7 + \frac{1}{d_7} * 0.7 + \frac{1}{d_4} * 0.4 + \frac{1}{d_8} * 0.4 \\ &= \frac{1}{D(g_4, G_6)} + \frac{0.7}{D(g_4, G_5)} + \frac{0.7}{D(g_4, G_7)} + \frac{0.4}{D(g_4, G_4)} + \\ &\quad \frac{0.4}{D(g_4, G_8)} \\ &= \frac{1}{16} + \frac{0.7}{20} + \frac{0.7}{16} + \frac{0.4}{20} + \frac{0.4}{14} \approx 0.19 \end{aligned}$$

其中, D 为距离函数,如上文所示。类似地,可以计算其他可用时隙的重用因子: R_3 换0.17, R_5 换0.18, R_8 换0.22, R_9 换0.19。于是,本文算法选择 t_8 来调度 A_4 。

4.2.2 算法扩展

基本算法假设所有数据访问的长度为1。然而,在实际应用中,数据访问需要多次迭代才能完成,具体取决于请求的数据量和访问存储了这些数据的I/O节点所需时间。如果数据访问有多次迭代,我们在计算重用因子时,将每次数据访问分成多个部分(子访问),进而对基本算法做少量改进。

更具体地,扩展算法首先根据数据访问的松弛度对数据访问进行排序,然后从松弛度最小的访问开始。为了确定每次访问的调度点,它检查松弛度内的所有时隙,如基本算法一样,排除不可用时隙。然而,为了计算数据访问松弛度内所有可用时隙的重用因子,扩展算法将所有其他数据访问分解为多个单位数据访问,每个单位数据访问假设需要一个时隙,然后根据这些单位数据访问,在目标时隙的垂直重用范围内计算重用因子。由于数据访问现在具有了长度,因此数据访问目标时隙 t 的垂直重用范围将是 $t-\delta$ 和 $t+l+\delta$ 间的迭代数量,其中 l 是该次访问的长度, δ 是基本算法的预设值。

我们现在假设一个例子,具有不同长度的5次数据访问,如图8所示,以给出如何用扩展算法来计算重用因子。

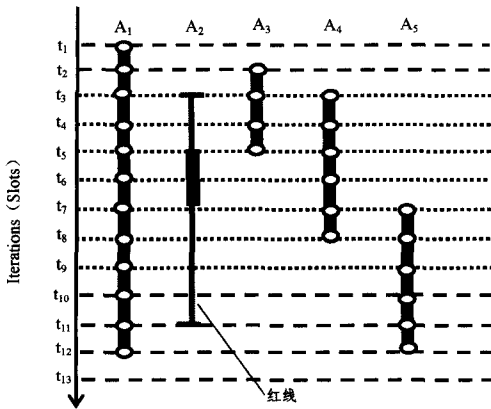


图8 长度大于1时数据访问调度示例

假设有4次访问(A_1, A_3, A_4, A_5)的调度点已经确定。尤其地,长度为12的 A_1 已经调度到 t_1 ,长度为4的 A_3 已经调度到 t_2 ,长度为6的 A_4 调度到 t_3 ,长度为6的 A_5 调度到 t_7 。现在,我们希望确定长度为3的第2次访问(A_2)的调度点,其松弛度在图8中用红线表示(从 t_3 到 t_{11})。假设我们考察可用时隙 t_5 是否适合该次访问,且垂直可用范围(δ)为2。为了

计算 t_5 的重用因子,我们将所有其他访问分解为单位数据访问,如图8中的虚线圆所示。各个单位数据访问的签名与其原先数据访问相同。然后,我们在 t_5 的垂直可用范围内,对单位数据访问使用式(1)和式(2)。由于 A_2 的长度为3,因此 t_5 的垂直可用范围包括 t_3 到 t_7 之间的迭代。尤其地, t_5, t_6, t_7 的权重为1, t_4, t_8 的权重为0.7, t_3 和 t_9 的权重为0.4。以下等式给出了 A_2 相对 t_5 的重用因子计算方法:

$$\begin{aligned} \mathcal{R}_5 &= \frac{1}{d_5} * 1 + \frac{1}{d_6} * 1 + \frac{1}{d_7} * 1 + \frac{1}{d_4} * 0.7 + \\ &\quad \frac{1}{d_8} * 0.7 + \frac{1}{d_3} * 0.4 + \frac{1}{d_9} * 0.4 \\ &= \frac{1}{D(g_2, G_5)} + \frac{1}{D(g_2, G_6)} + \frac{1}{D(g_2, G_7)} + \frac{1}{D(g_2, G_4)} \end{aligned}$$

其中, D 是距离函数, g_i 是数据访问 A_i 的签名,根据迭代 t_i 处的单位数据访问来确定群组活跃签名 G_i ,比如 $G_5 = g_1 | g_3 | g_4, G_6 = g_1 | g_4$ 。

4.2.3 磁盘性能讨论

磁盘空间周期变长后,磁盘在低功耗模式和活跃模式间的切换次数变少,因此磁盘空间周期变长对磁盘性能有潜在的积极性影响,但是上文描述的算法在设计时只是着眼于I/O节点节能最大化,没有明确考虑磁盘性能。我们发现,在部分情况下,如果数据访问调度时节能效果过于明显,可能会降低I/O性能。原因是,调度可能会在短时间内把大量数据访问放入同一个I/O节点,进而导致较长的排队等待,降低了I/O子系统的并行性能。

为了避免系统性能显著下降,我们在算法中引入参数 θ 作为阈值,以减少对各个I/O节点的一次性数据访问次数。例如,如果 θ 设为2,则对同一I/O节点,最多有两次访问(来自不同进程)在同一时隙内调度。我们首先给出如何向4.2.1节中的基本算法加入该性能约束条件。我们不是选择重用因子最高的时隙,而是首先根据它们的重用因子以非升序次序对所有可用时隙排序,然后依次检查这些时隙,直到找到一个点(时隙),所有I/O节点满足 θ 约束。如果没有时隙满足该约束,则我们选择对I/O节点的平均额外访问次数最少的时隙。假设 $\mathcal{Q}_t$ 是在时隙 t 时数据访问数量超过 θ 个的节点集合, M_d 表示对节点 d 的访问数量,于是,时隙 t 时平均额外访问数量 E_t 可计算为:

$$E_t = \frac{\sum (M_d - \theta)}{|\mathcal{Q}_t|}, d \in \mathcal{Q}_t$$

对于第4.2.2节描述的扩展算法,只有从 t 到 $t+l$ 间的所有迭代满足 θ 约束时,时隙 t 才能成为数据访问的调度点,其中 l 是数据访问的长度。此外,为了计算时隙 t 时对 g_i I/O节点的平均额外访问次数,我们需要考察 t 到 $t+l$ 间的所有迭代。让我们再次以图8为例,假设图中的数据访问发送给带有4个节点的I/O架构,且它们的签名 g_i 见表1。如果 $\theta=2$,则时隙 t_5 是合格调度点,因为根据已知的签名,对 t_5 到 t_7 间的每个时隙,针对同一I/O节点的数据访问数量均大于2。

表1 图8中数据访问的签名

g_1	g_2	g_3	g_4	g_5
0110	0100	0010	0001	0010

5 实验评估

5.1 设置和方法

我们使用文献[21]中的Phoenix编译器架构来部署本文

基于编译器的数据访问调度方案。实验中记录的最长编译时间约为 1.4 秒,比不使用本文方法时编译时间要长 40% 左右。我们还使用文献[22]中的 AccuSim 模拟器来模拟双层存储缓存架构,该架构包括客户端和服务节点,且每个服务器节点维护一个支持 I/O 预取的存储缓存。本文 I/O 栈包括基于文献[23]PVFS 并行文件系统的 MPI-IO。为了准确模拟连接到 I/O 节点的磁盘,AccuSim 与 DiskSim 通过接口进行通信,DiskSim 提供了大量计时和配置参数以便为 I/O 接口明确磁盘、控制器和总线。我们还向 DiskSim 加入了详细的功率模型,以便记录不同的磁盘操作/状态(比如搜索、旋转、读取/写入(使用多速磁盘时在不同速度下的读取/写入)和空闲时)期间的功耗统计数据。同时,我们的多速磁盘部署模型也详细模拟了磁盘 RPM 变化时导致的排队和服务延时。主要配置参数及其默认取值如表 2 所列。然后我们更改部分参数的默认值,以进行敏感度分析。

表 2 本文实验的主要参数

参数	数值
降速磁盘和多速磁盘的共用参数	
客户端(计算)结点数量	32
I/O 结点数量	8
数据块尺寸	64kB
RAID 等级	5,10
各个磁盘的容量	100GB
存储缓存容量	64MB(每个 I/O 结点)
最大磁盘转速	12000RPM
空闲功率	17.1W(12000RPM 转速时)
活跃(R/W)功率	36.6W(12000RPM 转速时)
搜索功率	32.1W(12000RPM 转速时)
待机功率	7.2W
提速功率	44.8W
提速时间	16secs
降速时间	10secs
磁盘臂调度	升降器(elevator)
总线类型	Ultra-3 SCSI
多速磁盘参数	
功率模型类型	二次方(见式(1))
最小磁盘转速	3600RPM
RPM 步进量	1200
算法参数	
δ	20 次迭代(时隙)
θ	4

以部分初步实验为基础,使用简单的节能策略时,我们设置磁盘降速到 50 毫秒(此时,对所有应用的节能效果较好,且将性能开销约束在 15% 以下)前的时间段为等待时间。在预测策略下,当空闲状态被预测到时,我们选择相应的速度以将观察到的最大性能损失降低到 4% 以下。在历史策略下,对每个应用,我们选择合适的 RPM 水平,将性能开销约束在 4% 左右。在交错策略下,我们在进入下一更低速度时等待 50 毫秒。同时,我们将 δ 默认值设为 20 次循环,将 θ 默认值设为 4。

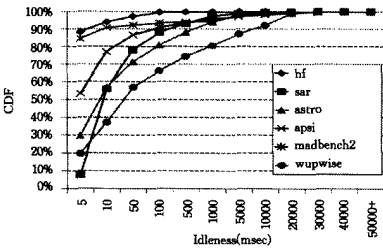
表 3 给出了本文研究时用到的一组并行 I/O 密集型应用,这些应用体现了各种数据访问模式。这些应用处理的磁盘数据总体规模的范围为 189.6GB (sar)到 446.1GB (wupwise)之间。第 3 列和第 4 列给出了没有采用节能策略时的运行时间和磁盘能耗,本文中称为默认方案。第 5.2 节和 5.3 节给出的能耗和性能衰减数据均是相对于默认方案而言的。

表 3 应用程序

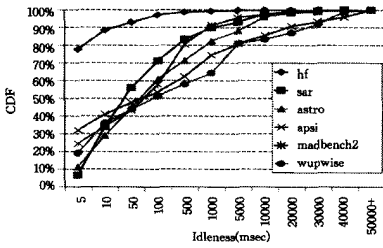
名称	简要描述	运行时间 (分钟)	磁盘能耗 (焦耳)
hf	Hartree-Fock 算法	27.9	3637.4
sar	合成孔径雷达内核	11.1	1227.3
astro	天文数据分析	16.8	2837.6
apsi	污染分布建模	13.7	3094.1
madbench2	背景辐射计算	9.8	1955.3
wupwise	物理/量子染色体动态	39.8	4812.1

5.2 没有使用本文方案时的结果

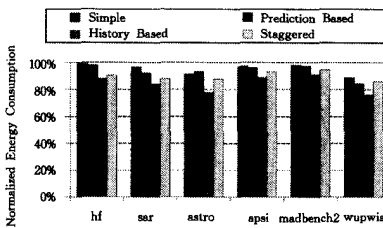
首先在图 9(a)中给出没有使用本文策略时,各种应用的磁盘空闲周期长度的累积分布函数(CDF)。具体地,该图中的 $(x, y\%)$ 表示磁盘空闲周期中有 $y\%$ 部分的长度小于等于 x 毫秒。在 hf 和 madbench 等应用中,空闲周期非常短(90% 以上的空闲周期低于 50 毫秒),因此难以使用传统的降速策略。可以看出,平均来讲,86.4% 的空闲周期的长度小于等于 100 毫秒,96.5% 的长度小于等于 5 秒。图 9(c)给出了没有使用本文策略时,4 种磁盘节能方案的正规化能耗值。如我们预期,结果并不理想。相比之下,预测策略的性能较优,与默认方案相比,节约了 6.3% 左右。对于多速磁盘策略,我们发现,历史策略的节能率为 15.6%,交错策略的平均节能率为 9.8%。总体来说,历史策略的节能效果优于其他策略。



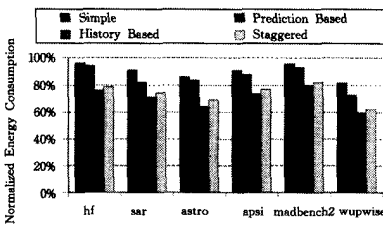
(a) 未用本文算法时空闲周期的 CDF 函数



(b) 使用本文算法时空闲周期的 CDF 函数



(c) 未用本文算法时归一化能耗



(d) 使用本文算法时归一化能耗

图 9 不同方案的 CDF 和能耗对比

以上策略的性能特征如图 10(a)所示。Y 轴给出了相对默认方案的性能衰减。可以发现,原始策略的性能衰减非常严重(平均约 10.4%)。主要原因是,在运行期间空闲周期的长度发生变化时,该方案无法做出调整。许多情况下,空闲周期较短,调高速度会对性能产生较大影响。为了解决这一问题,有人建议采用更长的等待周期。然而,使等待周期加倍会显著降低性能衰减,同时节能率不到 1%,因为我们无法找到机会降低磁盘速度。预测策略和历史策略可以预测下一空闲周期,因此性能衰减较低。另一方面,交错策略的性能衰减较为严重,因为该策略下下一次数据访问可能发生于磁盘刚好调整到低速模式时,此时,对应的恢复时间会较长。

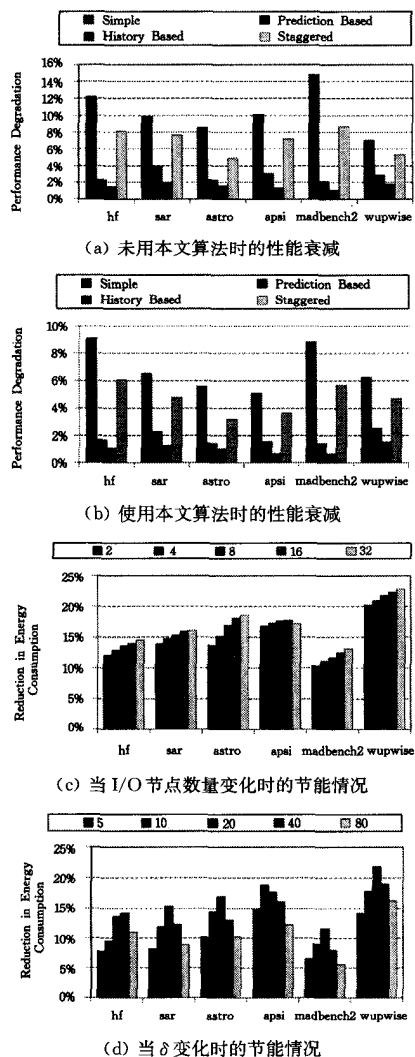


图 10 性能衰减和节点情况分析

5.3 使用本文方法时的结果

本文提出的数据调度方法有助于提高底层功率功能策略的效益。现在,我们对这一效益进行量化。图 9(b)给出了使用本文方法时磁盘空闲周期的 CDF 函数。可以看出,空闲时间有显著提升。例如,在图 9(a)中,长度小于等于 500 毫秒的空闲周期比例达到 90.4%,在图 9(b)中这一比例达到 75.7%。图 9(d)和 10(b)分别给出了空闲周期增加后对能耗和性能的影响。从图 9(d)可以看出,使用本文方法后,4 种策略的节能率分别达到 9.4%,14.2%,29.2%和 25.9%,远优于不使用本文方法(4.7%,6.3%,15.6%和 9.8%)。因此,我们认为本文方法在实践中非常有效。此外,图 10(b)中的性

能结果表明,本文方法也有助于性能提升。例如,原始策略的平均性能衰减率从 10.4%下降到 6.9%;类似地,历史策略的平均性能衰减率从 1.5%下降到 1%。

5.4 敏感性结果

我们还通过改变重要参数的默认值来进行敏感性研究。在下面给出的每个实验中,每次更改一个参数,其余参数保持不变(取表 2 中的值)。由于各种策略的趋势类似,因此我们重点讨论历史策略及使用本文方法后的额外节能效果。

我们实验的每一个参数是 I/O 节点数量,结果如图 10(c)所示(默认值为 8)。我们发现,当 I/O 节点数量增加时,总体来说节能效果也会增加。但是,上升效果不明显。原因是,这些结果是相对于历史策略而言的,而历史策略已经利用了 I/O 节点的数量上升。然后,我们改变 δ 参数的值。图 10(d)给出了保持第 4.2.2 节加权策略不变的同时改变 δ 值时的结果。我们发现, δ 值过大或过小会降低我们潜在的节能效果(原因不同)。具体来说,当 δ 较大时,算法会错误地假设磁盘仍然处于活跃状态,然后企图重用磁盘。这可能会导致部分磁盘从低功耗模式转到活跃模式,进而降低我们的节能效果。另一方面,如果 δ 较小,则会错误假设部分活跃磁盘处于低功耗模式,进而降低了本文方法调度数据访问时的灵活性,影响了节能效果。如何选择合适的 δ 值也是个重要课题,我们下一步将对此展开研究。图 11(a)给出了 θ 取不同值时的节能效果。请注意, θ 值限制了在已知调度时隙访问同一 I/O 节点的客户端节点数量,且默认值为 4。我们从图 11(a)可以看出,与我们预期一致,取值较大时会增加我们的节能效益,但也会影响性能,如图 11(b)所示。与 δ 参数一样, θ 值的取值问题也在我们下步研究计划之内。我们还改变了 I/O 节点的存储缓存容量,并且发现,当把缓存容量从 64MB 下降到 32MB 时,历史策略使用本文方法后节能效果提升 4.3%左右。此外,当把缓存容量提升到 256MB 时,节能效果下降 3.7%。这是因为,存储缓存本身有助于降低磁盘的活跃性,缓存越大,它节省的磁盘访问次数越多,本文方法实现的效益越小。

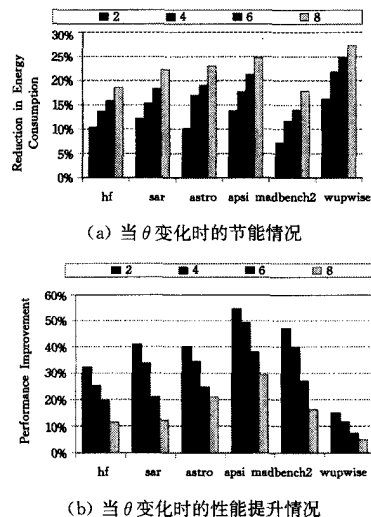


图 11 敏感性分析结果

结束语 本文的贡献在于提出一种基于编译器的数据访问调度策略,可实现并行应用程序空闲周期的最大化,进而降低磁盘能耗。该策略包含两个部分:进行数据访问调度的优化编译器,根据调度节点进行数据预取和缓存的运行期间调度器。本文方法可以避免大规模更改代码和数据布局。基于

(下转第 223 页)

- [M]. Beijing: Foreign Language Teaching and Research Press, 2009
- [2] 邱晗,周强. 自动获取大规模的汉语紧密组合同义关联对[J]. 清华大学学报:自然科学版,2011(9):28-33
- [3] 余一骄,刘芹. 面向超大规模的中文文本 N-gram 串统计[J]. 计算机科学,2014(4):263-268
- [4] 罗瑜昕. 用统计的方法看“京派”与“海派”小说语言风格差异[J]. 现代语文:学术综合版,2012(4):137-141
- [5] 陈功. 语料库检索的模式、问题及启示[J]. 当代外语研究,2011(10):10-14
- [6] 任海波. 现代汉语 AABB 重叠式词构成基础的统计分析[J]. 中国语文,2001(4):302-308
- [7] 崔四行. 从 ABAB、AABB 重音模式的句法功能看汉语的韵律形态[J]. 语言教学与研究,2012(5):63-69
- [8] 蒋向勇,白解红. 汉语 ABB 式网络重叠词语的认知解读[J]. 外语研究,2013(3):30-34
- [9] 邢福义. “X 以上”格式在现代汉语中的演进[J]. 语言研究,2010(1):1-10
- [10] 洪涛,董正存. “非 X 不可”的历史演化和语法化[J]. 中国语文,2004(3):253-261
- [11] 邵敬敏. 说“V 一把”中 V 的泛化与“一把”的词汇化[J]. 中国语文,2007(1):14-19
- [12] 李广瑜. 否定祈使句式“别 V 着”刍议[J]. 语言教学与研究,2013(1):48-55
- [13] 吴为善,夏芳芳. “A 不到哪里去”的构式解析、话语功能及其成因[J]. 中国语文,2011(4):326-333
- [14] 张谊生. “N”+“们”的选择限制与“N 们”的表义功用[J]. 中国语文,2001(3):201-211
- [15] <http://ccl.pku.edu.cn/8080>
- [16] <http://democlip.blcu.edu.cn/800>
- [17] 王惠. 词义·词长·词频——《现代汉语词典》(第 5 版)多义词计量分析[J]. 中国语文,2009(5):120-130
- [18] 张宝林. 汉语中介语语料库建设的现状与对策[J]. 语言文字应用,2010(3):129-138
- [19] Zhang Hua-ping, Yu Hong-kui, Xiong De-yi, et al. HHMM-based Chinese Lexical Analyzer ICTCLAS [C]//Proceedings of 2nd SIGHAN Workshop Affiliated with 41st ACL, 2003:184-187

(上接第 197 页)

6 种应用程序的实验表明,本文方法可以有效降低磁盘能耗。在下一步研究工作中,我们将研究多应用背景下的磁盘空闲周期增加问题。

参 考 文 献

- [1] 易会战,罗兆成. 基于动态电压调节的高性能业务系统能耗优化[J]. 华中科技大学学报:自然科学版,2013(1):25-29
- [2] 张凯,陈书明,王耀华,等. 面向通用 HPC 的高性能 DSP 设计权衡[J]. 计算机学报,2013,36(4):790-798
- [3] 张帅,宋凤龙,王栋,等. 多核结构片上网络性能-能耗分析及优化方法[J]. 计算机学报,2013,36(5):988-1003
- [4] Hadjipaschalis I, Poullikkas A, Efthimiou V. Overview of current and future energy storage technologies for electric power applications[J]. Renewable and Sustainable Energy Reviews, 2009,13(6):1513-1522
- [5] Li K, Kumpf R, Horton P, et al. A quantitative analysis of disk drive power management in portable computers[C]//USENIX winter, 2002:279-291
- [6] Gurumurthi S, Sivasubramaniam A, Kandemir M, et al. DRPM: dynamic speed control for power management in server class disks[C]//30th Annual International Symposium on Computer Architecture, 2003. IEEE, 2003:169-179
- [7] Son S W, Chen G, Kandemir M, et al. Exposing disk layout to compiler for reducing energy consumption of parallel disk based systems[C]//Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM, 2005:174-185
- [8] Son S W, Kandemir M, Choudhary A. Software-directed disk power management for scientific applications[C]//19th IEEE International Parallel and Distributed Processing Symposium, 2005. IEEE, 2005:4-13
- [9] 李元章,孙志卓,马忠梅,等. S-RAID5:一种适用于顺序数据访问的节能磁盘阵列[J]. 计算机学报,2013,36(6):1290-1302
- [10] 刘靖宇,郑军,李元章,等. 混合 S-RAID:一种适于连续数据存储的节能数据布局[J]. 计算机研究与发展,2013,50(1):37-48
- [11] 孟涛,刘浩,胡宏扬. 基于预读策略的节能数据访问技术[J]. 计算机工程,2012,38(8):44-46
- [12] 李建敦,彭俊杰,张武. 云存储中一种基于布局的虚拟磁盘节能调度方法[J]. 电子学报,2012,40(11):2247-2254
- [13] Weissel A, Beutel B, Bellosa F. Cooperative I/O: A novel I/O semantics for energy-aware applications [J]. ACM SIGOPS Operating Systems Review, 2002,36(SI):117-129
- [14] Papathanasiou A E, Scott M L. Energy efficient prefetching and caching[C]//Proceedings of the 2004 USENIX Annual Technical Conference. Berkeley, CA, USA, 2004:255-268
- [15] Pinheiro E, Bianchini R. Energy conservation techniques for disk array-based servers[C]//Proceedings of the 18th annual international conference on Supercomputing. ACM, 2004:68-78
- [16] Son S W, Kandemir M, Choudhary A. Software-directed disk power management for scientific applications[C]//19th IEEE International Parallel and Distributed Processing Symposium, 2005. IEEE, 2005:4b-4b
- [17] Son S W, Kandemir M. Energy-aware data prefetching for multi-speed disks[C]//Proceedings of the 3rd Conference on Computing Frontiers. ACM, 2006:105-114
- [18] Thakur R, Gropp W, Lusk E. Data sieving and collective I/O in ROMIO[C]//The Seventh Symposium on the Frontiers of Massively Parallel Computation, 1999 (Frontiers'99). IEEE, 1999:182-189
- [19] Liao W, Coloma K, Choudhary A, et al. Collective caching: application-aware client-side file caching[C]//14th IEEE International Symposium on High Performance Distributed Computing, 2005 (HPDC-14). IEEE, 2005:81-90
- [20] Calder P C, Jacobsen C, Skall Nielsen N, et al. Nutritional benefits of omega-3 fatty acids[M]//Food enrichment with omega-3 fatty acids. 2013:3-26
- [21] Whaley J. Joeq: A virtual machine and compiler infrastructure [C]//Proceedings of the 2003 Workshop on Interpreters, Virtual Machines and Emulators. ACM, 2003:58-66
- [22] Palmer A J, Brandt A, Gozzoli V, et al. Outline of a diabetes disease management model: principles and applications [J]. Diabetes research and clinical practice, 2000,50:S47-S56
- [23] Ligon W, Ross R. PVFS: Parallel virtual file system[M]//Beowulf cluster computing with Linux. MIT Press, 2001:391-429