桶外排序算法的抽样分点分发策略*

杨 磊[1+], 黄 辉[2], 宋 涛[1]

[1](清华大学 计算机科学与技术系,北京 100084)

[2](中联绿盟信息技术(北京)有限公司 开发部,北京 100089)

The Sample-Seperators Based Distributing Scheme of the External Bucket Sort Algorithm

YANG Lei[1+], HUANG Hui[2], SONG Tao[1]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](Department of Development, NSFOCUS Information Technology Co., Ltd., Beijing 100089, China)

+ Corresponding author: Phn: +86-10-62777918, E-mail: yanglei@sheenk.com, http://www.tsinghua.edu.cn

Received 2004-03-23; Accepted 2004-06-11

Yang L, Huang H, Song T. The sample-seperator based distributing scheme of the external bucket sort algorithm. *Journal of Software*, 2005,16(5):643−651. DOI: 10.1360/jos160643

Abstract: Two ways to sort externally are Multi-Line Merging Sort and Bucket Sort, both with two passes. The Bucket Sort burdens the CPU less and is more efficient, while its usage is restricted heavily by the High-Bit scheme that distributes records into subfiles: the keys have to be integers; the sizes of subfiles may vary too much; the number of subfiles cannot be chosen freely. Based on statistical theory, this paper presents a sample-seperators scheme to broaden the ussage of bucket sort algorithm. A brief discussion on the convergance of sample-seperator estimation is given and the probability to avoid memory overflow is calculated. This scheme enables the bucket sort algorithm to be applied in the SheenkSort system to win the 2003 PennySort (the Indy category) competition.

Key words: external sort; bucket sort; multi-line merging; distributing scheme; sample-separators; PennySort

摘 要: 计算机外排序常用二阶段多路归并算法和桶算法.后者运算开销小,效率更高.但基于关键字高位比特的子文件分发策略应用受限:关键字必须是整数;得到的子文件可能大小不一;子文件数不能任意选择.基于统计学理论,提出抽样分点分发策略克服以上问题,扩展桶排序的应用范围.讨论了抽样分点估计的收敛性,给出了不发生内存溢出的保证概率.该策略使桶排序算法在 SheenkSort 排序系统上得到成功应用,并最终获得 2003 年度 PennySort 世界排序比赛 Indy 组冠军.

关键词: 外排序;桶排序;多路归并;分发策略;抽样分点;PennySort

中图法分类号: TP301 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60223004, 60321002, 60303005 (国家自然科学基金)

作者简介: **杨磊**(1978−),男,湖北武汉人,博士生,主要研究领域为计算机人工智能;**黄辉**(1978−),男,主要研究领域为计算机网络安全及应用;**宋涛**(1978−),男,博士生,主要研究领域为计算机信息安全.

排序问题是计算机领域十分基础而重要的问题.针对小规模数据量,人们已经提出了各种各样的内排序算法,例如冒泡排序[1]、快速排序[2]等,以及近几年提出的分段快速排序[3]、基数链接排序[4]、分片扩展排序[5]等.但当数据规模超过了内存限制,就必须采用一定的外排序算法,将待排数据分片读入内存进行处理[6,7].常采用二阶段多路归并算法和桶排序算法.等量数据下,后者分发操作比前者的归并操作要简单,占用资源较少,有利于算法效率的提高.

但是,桶排序算法的应用要求找到一种简单的由关键字到所属子文件的映射关系.高位策略取整型关键字的若干高位比特以确定子文件编号,但这在很多情况下行不通.首先,无法处理非整型关键字(例如浮点型).其次,对于非均匀分布数据所得到的子文件大小可能相差很大,内存利用率低.再次,最优子文件数 M 由问题本身决定,但高位策略中 M 却由关键字本身的形式所决定,这使得算法往往达不到最优的效率.由于上述原因,桶排序算法难以得到广泛应用.

注意到,待排记录一般可以认为是独立同分布的.基于统计学理论,本文提出一种通过抽样等分点估算桶边界的子文件编号策略,以克服上述问题,使桶排序算法能够在更一般的情形下得到应用.对于概率算法,本文从理论上推导了算法在一定样本容量下保证不发生内存溢出的概率,并讨论了抽样分点估计的收敛性.

外排序任务涉及计算机各组成部分的协同工作能力,是衡量系统的综合性能的一个良好标准.基于此,1998年度图灵奖获得者 Jim Gray 先生倡导了包括 PennySort 在内的一系列排序比赛[8],旨在追踪世界最新的计算机软、硬件发展水平.本文提出的抽样分点策略使桶排序算法在 PennySort 任务上的应用成为可能,并据此实现了 SheenkSort 排序系统,以 1541s 处理 42GB 数据的成绩获得 2003 年度 PennySort 比赛 Indy 组冠军[9].本文仔细分析了 PennySort 数据特征,指出其无法应用高位策略,却可以使用抽样分点策略.而与使用多路归并算法的 THSort 系统[10]性能对比,则显示出桶排序算法的优势.

1 二阶段多路归并算法

假定原始待排文件所包含的记录总数为 N,算法分两步进行.第 1 步,排序阶段:将 N 条待排记录均匀分成 M 个子文件,每个子文件中包含 $R=N/M$ 条记录.在内存中分别对每个子文件进行内排序得到 M 个有序的子文件.第 2 步,归并阶段:将这 M 个有序子文件合并成一个完整的有序文件,即归并.这只需要从 M 个子文件头记录(也就是该子文件中当前最小的记录)依次选出最小者即可.如图 1 所示.

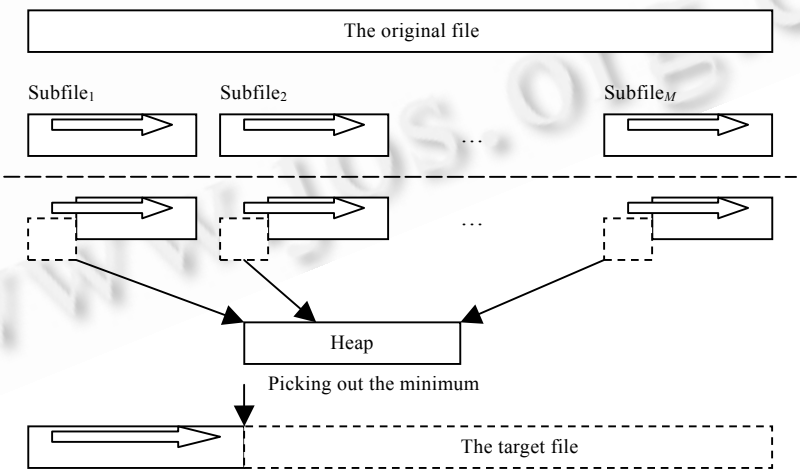

Fig.1 The multi-line merging algorithm (above the dashed line is the sorting phrase and below is the merging phrase)
图 1 多路归并算法(虚线上方为排序阶段,下方为归并阶段)

排序阶段一次从原始文件中读入 R 个记录进行内排序,然后写回相应子文件.排序阶段结束后得到 M 个排好序的子文件.这一阶段系统的磁盘开销包括将所有数据读入和写出各 1 次,CPU 开销则来自于对 M 个文件进

行内排序,后者复杂度为 $M\cdot O(R\log R)=O(N\log R)$.可以看到,$R$ 越大 M 越小,对算法越不利.归并阶段的磁盘开销同样包括将所有数据读入和写出各 1 次,而 CPU 开销则主要来自于归并操作.一种比较高效的归并算法是通过“堆”这种二叉树的数组表示[11],能够在 $O(\log M)$时间内挑选出 M 个头记录中的最小者.总的时间复杂度为 $N\cdot O(\log M)=O(N\log M)$,$M$ 越大,对算法越有利.

考虑二阶段算法的综合性能,我们需要根据实验来确定最佳的子文件数 M.这里,算法对 M 取值的限制是 $R=N/M$ 不能超过内存容量,否则会导致子文件的内排序失败(内存溢出).

2 桶排序算法

桶排序算法也分成两步进行.第1步,分发阶段:首先根据一种编号策略将所有记录分发到M个子文件(形象地称为“桶”),使得一个子文件中的所有记录都比其前一个子文件中的所有记录要大.第 2 步,排序阶段:依次对这 M 个子文件进行排序.将这 M 个有序的子文件连接起来便得到了最终的排序结果.如图 2 所示.

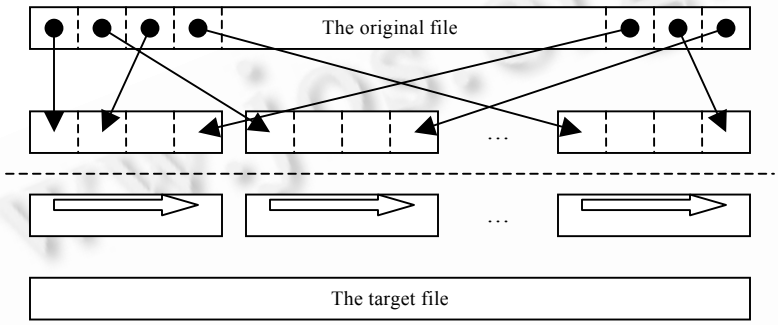

Fig.2 The bucket sort algorithm (above the dashed line is the distributing phrase and below is the sorting phrase)
图 2 桶排序算法(虚线上方为分发阶段,下方为排序阶段)

桶排序的第 2 步和多路归并的第 2 步非常相似,都是对 M 个子文件进行排序——两种算法在这一步的系统开销基本一样.区别在于,多路归并算法的第 2 步中包含复杂的归并过程,而对于桶算法,只要分发策略足够简单,其分发阶段的 CPU 开销可以非常小.例如,当关键字由一个 4 字节整数构成,而我们希望把所有记录放入 256 个子文件中时,我们可以仅根据关键字的第 1 字节确定应该将其放入哪一个子文件.第 1 个子文件中的记录关键字第 1 字节都等于 0,而第 2 个子文件中的关键字第 1 字节都等于 1;后者中的任一记录显然要大于前者中的所有记录.对于这样的分发策略,分发阶段的 CPU 开销为线性(几乎等同于文件拷贝),低于多路归并算法第 2 步中的归并开销.这种策略也是当前多数桶排序算法在实现中所采取的分发策略.

遗憾的是,很多情况下,这样的高位策略并不可行甚至根本不存在.首先,如果关键字不是整型,无法仅根据高位比特判断大小(如浮点型关键字),类似的分发策略便不存在.其次,桶排序算法中的 R 只是所有子文件平均所包含的记录数,我们无法预知最终会有多少记录落入某个子文件中.内存必须能够容纳最大子文件,而它所包含的记录数可能会比 R 大很多.实际实现中,M 常取得很大以避免内存溢出,极大地降低了算法效率.另外,高位策略中的子文件数 M 由关键字具体形式决定,不一定适合我们的算法.因此,桶排序算法只在很特殊的条件下才能够得到应用.

3 抽样分点分发策略

综上我们需要这样一种分发策略:不依赖于关键字的具体形式,可以处理整型以外的关键字;能够保证得到大小接近的子文件;子文件数 M 可以任意调整以适应算法的需要.注意到子文件 $\mathbf{F}_k$ 其实仅由其最大和最小的两条记录 $x_k^{\min}$ 和 $x_k^{\max}$ 所确定:凡介于二者之间的记录都应置于子文件 $\mathbf{F}_k$ 中.于是只要确定了边界点集 $\mathbf{X}^{(M)}=\{x^0, x^1,\ldots,x^M\}$,我们就得到了完整的分发策略:对于任何 x 只要找到 k 使得 $x^k\le x<x^{k+1}$,那么 x 应置于 $\mathbf{F}_k$ 内,而这可以通

过二分查找很容易做到的.剩下的便是如何确定边界点集 $\mathbf{X}^{(M)}$.

理想情况下,我们希望所得到的子文件大小相等,这意味着 $\mathbf{X}^{(M)}$就是待排数据集的 M 均匀等分点(即 $1/M$, $2/M,\ldots,(M-1)/M$ 分点,以及最大、最小共 $M+1$ 个点).要精确找到这些分点并不容易,一种近似的方法是用一个较小且易于处理样本集分点作为其估计值.该策略确实不依赖于关键字的具体形式,子文件数 M 也可在一定范围内任意选择,但能够保证最终所得到的文件大小接近么?

在实际的排序问题中,记录往往可以认为是独立同分布的,记分布函数为 $D(x)=Pr(X\leq x)$,并假设其连续.对于任意样本记录 x,它应该是全数据集的 $p=D(x)$分点.设 $p\in(0,1)$及$\alpha\in(0,1)$,记 $q=1-p,\beta=1-\alpha$,对于包含 S 个记录的样本集,则 x 成为其α分点(正好有αS 个记录比 x 小,另βS 个记录比 x 大)的概率为

$$\begin{aligned}P_{(\alpha)}^{p} &= \frac{S!}{(\alpha S)!(\beta S)!}p^{\alpha S}q^{\beta S}\\ &\approx \frac{\sqrt{2\pi S}\left(\frac{S}{e}\right)^{S}}{\sqrt{2\pi\alpha S}\left(\frac{\alpha S}{e}\right)^{\alpha S}\sqrt{2\pi\beta S}\left(\frac{\beta S}{e}\right)^{\beta S}}p^{\alpha S}q^{\beta S}\\ &= \frac{1}{\sqrt{2\pi\cdot\alpha\beta\cdot S}}\alpha^{-\alpha S}\beta^{-\beta S}p^{\alpha S}q^{\beta S}\\ &= \frac{1}{\sqrt{2\pi\cdot\alpha\beta\cdot S}}\left(\left(\frac{p}{\alpha}\right)^{\alpha}\left(\frac{q}{\beta}\right)^{S}\right).\end{aligned}$$

其中用到了 Stirling 公式:当 n 足够大时 $n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}$.注意到,$\left(\frac{p}{\alpha}\right)^{\alpha}\left(\frac{q}{\beta}\right)^{\beta}\leq 1$ 且仅当 $p=\alpha$(同时有 $q=\beta$)时取等号,而当$\left(\frac{p}{\alpha}\right)^{\alpha}\left(\frac{q}{\beta}\right)^{\beta}<1$时, $P_{(\alpha)}^{p}$ 会随着 S 的增大迅速衰减到 0.也就是说,当 S 足够大时,只有当 p 和α足够接近时,x 才有可能成为样本集的α分点.于是,样本集的α分点 x^{α}满足 $Pr(X\leq x^{\alpha})\approx\alpha$,$x^{\alpha}$可以看做是整个待排数据集的$\alpha$分点.

另外,我们还可以证明给定某子文件边界 $x^{\min}$ 和 $x^{\max}$ 记 $p=Pr(x^{\min}\leq X<x^{\max})$,则落入该子文件中的记录数服从参数为$\lambda=pN$ 的泊松分布.

$$\begin{aligned}Pr(\#\{x|x^{\min}\leq X<x^{\max}\}=n) &= \frac{N!}{n!(N-n)!}p^{n}(1-p)^{N-n}\\ &\approx \frac{\sqrt{2\pi N}\cdot\left(\frac{N}{e}\right)^{N}p^{n}(1-p)^{N-n}}{n!\cdot\sqrt{2\pi\pi N-n)}\cdot\left(\frac{N-n}{e}\right)^{N-n}}\\ &= \frac{\sqrt{\frac{N}{N-n}}\cdot\left(\frac{N}{N-n}\right)\cdot\left(\frac{N}{e}\right)p^{n}(1-p)^{N-n}}{n!}\\ &\approx \frac{\left(\frac{(1-p)N}{N-n}\right)^{N-n}\cdot\left(\frac{pN}{e}\right)^{n}}{n!}\\ &= \frac{\left(\frac{N-\lambda}{N-n}\right)^{N-n}\cdot\left(\frac{\lambda}{e}\right)^{n}}{n!}\\ &\approx \frac{e^{n-\lambda}\cdot\left(\frac{\lambda}{e}\right)^{n}}{n!}=\frac{\lambda^{n}}{n!}e^{-\lambda}.\end{aligned}$$

其中再次用到了 Stirling 公式.在桶排序算法实现中,设内存能够容纳的最大子文件所包含的记录数不超过 $R_{\max}$,则该子文件发生溢出的概率为

$$\begin{aligned}\sum_{r=R_{\max}+1}^{+\infty}\left(\frac{\lambda^r}{r!}\cdot e^{-\lambda}\right) &= e^{-\lambda}\cdot\sum_{r=\lambda+\delta+1}^{+\infty}\frac{\lambda^r}{r!}\\ &= e^{-\lambda}\cdot\sum_{r=\lambda+1}^{+\infty}\frac{\lambda^{r+\delta}}{r!\cdot(r+1)\cdot(r+2)\cdot\ldots\cdot(r+\delta)}\\ &< e^{-\lambda}\cdot\sum_{r=\lambda+1}^{+\infty}\frac{\lambda^{r+\delta}}{r!\cdot(\lambda+1)\cdot(\lambda+2)\cdot\ldots\cdot(\lambda+\delta)}\\ &< \frac{\lambda^{\delta}}{(\lambda+1)\cdot(\lambda+2)\cdot\ldots\cdot(\lambda+\delta)}\cdot e^{-\lambda}\cdot\sum_{r=1}^{+\infty}\frac{\lambda^r}{r!}\\ &= \frac{\lambda}{\lambda+1}\cdot\frac{\lambda}{\lambda+2}\cdot\ldots\cdot\frac{\lambda}{(\lambda+\delta)}.\end{aligned}$$

其中 $\delta=R_{\max}-\lambda$.仅考虑后 $\delta/2$ 项则不超过 $\left(\frac{\lambda}{\lambda+\delta/2}\right)^{\delta/2}=\left(\frac{2\beta}{1+\beta}\right)^{\frac{1-\beta}{2}R_{\max}}$,这里 $\beta=\lambda/R_{\max}\in(0,1)$ 为该子文件的内存利用率,$2\beta/(1+\beta)<1$,上述概率随 $R_{\max}$ 的增加而递减——扩大内存可以在更大程度上避免溢出的发生.另外,$\left(\frac{2\beta}{1+\beta}\right)^{\frac{1-\beta}{2}}$ 随 $\beta=\lambda/R_{\max}$ 的增加而递减,这表明在内存无法扩大的情况下我们应当通过减少 λ,也就是子文件中的期望记录数来避免溢出;而这是通过增加文件个数实现的.考虑全部 M 个子文件,记 β_m 为所有子文件内存利用率的最大值,则全部 M 个子文件均不发生内存溢出的概率不超过 $\left\{1-\left(\frac{2\beta_m}{1+\beta_m}\right)^{\frac{1-\beta_m}{2}R_{\max}}\right\}^M$.

举一个例子:将给定数据集分发到 $M=2$ 个子文件中只需要一个边界 x^{mid}:$\mathbf{F}_0=\{x\in\mathbf{X}|x<x^{\mathrm{mid}}\}$,$\mathbf{F}_1=\{x\in\mathbf{X}|x\geq x^{\mathrm{mid}}\}$,并且 x^{mid} 就是样本集的中位数.设数据服从标准正态分布($D(x)=\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}}\mathrm{d}t$),给定样本集{−1.43,1.81,−0.50,−0.11,−1.06,−1.51,0.59,0.21,0.52},中位数为 $x^{\mathrm{mid}}=-0.11$.设共有 $N=500$ 个数据待分发,$\lambda=N/M=250$.若 $R_{\max}=300$,则内存利用率为 $\beta=\lambda/R_{\max}=83\%$.代入上式,得到不发生溢出的保证概率为 82.4%,而这对应了 $\mathbf{F}_0/\mathbf{F}_1$ 中包含 200~300 个记录的情形,查表得 $Pr(x\in\mathbf{F}_0)=0.4462$,经计算得到不溢出的概率实际为 99.5%.

在抽样分点策略中,记录 x 所属子文件编号是通过在 $M-1$ 个边界中查找得到的.使用二分查找算法计算一个编号平均约需进行为 $O(\log M)$次比较,CPU 开销仍为 $O(N\log M)$,与多路归并的归并算法具有相同的阶,但系数却小很多:每处理一个记录二分查找仅涉及 $\log_2 M$ 次比较操作,而归并操作为了维持堆结构却需进行约 $2\log_2 M$ 次比较和 $\log_2 M$ 次交换.不过抽样分点策略需要事先对子文件边界进行估计,也就是要找到 S 个样本记录的 M 均匀等分点.如果 S 不超过内存限制,可将其读入内存,排序后直接取相应位置的分点.考虑到排序阶段要进行 M 次子文件内排序,这个子文件边界的估计过程运算开销不超过排序阶段的 $1/M$.当 M 较大时可不予考虑.

系统 I/O 操作需要占用一定的 CPU 资源.在 I/O 操作与运算过程并行的系统上一旦二者发生对 CPU 资源的争抢,将导致 I/O 效率受到严重影响,进而影响整个系统效率.桶排序算法 CPU 开销小,能为 I/O 操作节省更多的 CPU 资源,使系统性能得到更充分的发挥.

4 实 验

为模拟实际情况,标准 PennySort 记录由 10 字节关键字及 90 字节附加数据(共 100 字节)构成.关键字为整型,最高字节取值范围为 32~126 共 95 种取值.但是对于 SheenkSort 系统所处理的包含 433M 记录的待排数据集,实验发现,关键字在其第 1 字节上具有如图 3 所示的分布:记录数在第 1 字节取值等于 32 处出现了高达平均值两倍的尖锋,导致桶排序算法的内存利用率仅 50%(假定内存正好能够容纳最大的子文件).另外,95 个子文件

对于 433M 记录的排序任务又太少了(子文件太大),因此高位策略在这里并不合适(注:文中 1K=1024,1M=1024K).

抽样分点策略克服了上述问题.用前 70K 记录作为抽样样本,排序后取出其 300 等分点作为子文件边界,所得到的子文件大小分布如图 4 所示.经计算,内存利用率可以达到 84%.样本集分点通过内排序确定.由于样本集容量仅占全部数据集的 1/6000,其处理时间可以完全忽略不计.另外,这里的子文件数 300 也可以根据需要在一定范围内调整.

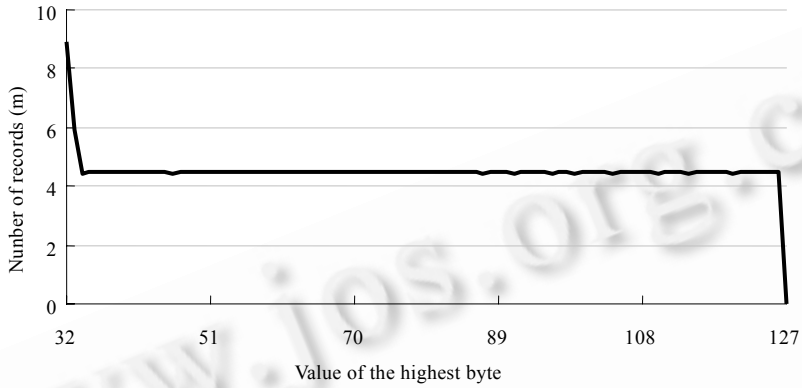

Fig.3 Distribution of PennySort records over the highest key byte (433M records)

图 3 PennySort 记录在关键字高字节下的分布(433M 记录)

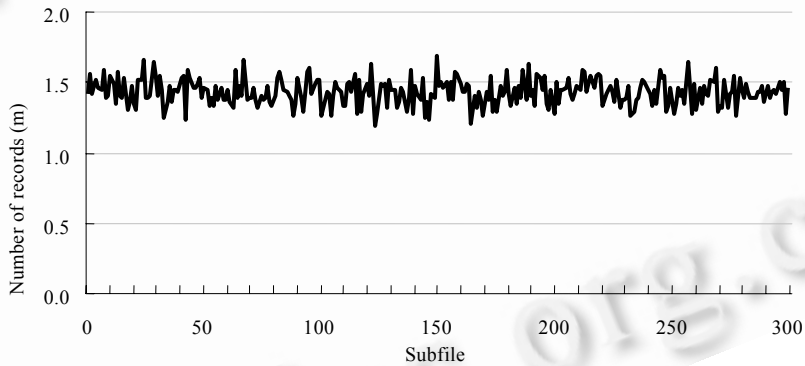

Fig.4 Size of each subfile under the sample-separator strategy

图 4 抽样分点分发策略下的各子文件中所包含的文件数

抽样分点分发策略使得桶排序算法在 PennySort 排序任务上的应用成为可能.基于本文思想所设计的 SheenkSort 排序系统在 2003 年度 PennySort 比赛上提交的最终结果为 1541s 内处理 42GB 数据(433M 记录,平均排序速率 28MB/s).其基本算法如下:

(1) 取待排数据集前 70K 记录作为样本集进行内排序,取其 300 等分点作为桶边界.

(2) 分发.根据第 1 步所确定的边界,依次将全部 433M 个记录分发到 300 个子文件中.实际使用两个并行进程,其中读进程将数据读入内存,而写进程则将数据分发到相应的临时文件中.

(3) 排序.依次将各临时文件读入内存排序后写回目标文件.依次处理完所有临时文件便得到有序的目标文件.采用我们自行编写的内排序算法,并使用了 3 个并行线程分别控制读、写、排序操作.

图 5 给出了 SheenkSort 排序系统在不同大小数据集下的实验结果.其中包括桶排序算法的分发阶段与排序阶段分别所消耗的时间.表 1 给出了系统硬件平台及 2002 年度 PennySort 比赛 Daytona 组冠军 THSort 的硬件配置.二者非常接近,这提供了将二者性能进行对比的可能.THSort 系统使用前文所述的多路归并算法[10],最后提交的结果为 1104s 处理 9.8GB 数据量,合 9MB/s,单位时间内所处理的数据量约为 SheenkSort 的三分之

一.THSort 系统的基本算法如下[10]:

(1) 排序.将数据顺序分成不超过内存大小的小块读入,对每一个小块进行排序,并将结果分别保存到临时文件中.采用我们自己实现的快速排序算法,并使用了 3 个并行进程分别控制读、写、排序操作.

(2) 归并.将第 1 步生成的所有临时文件进行归并排序,得到全局有序的数据.采用比较高效的堆排序算法.实际使用中使用两个并行进程,其中读进程将临时文件内的数据补充到内存中,而写进程则从内存中取出最小元素归并到输出目标文件中.

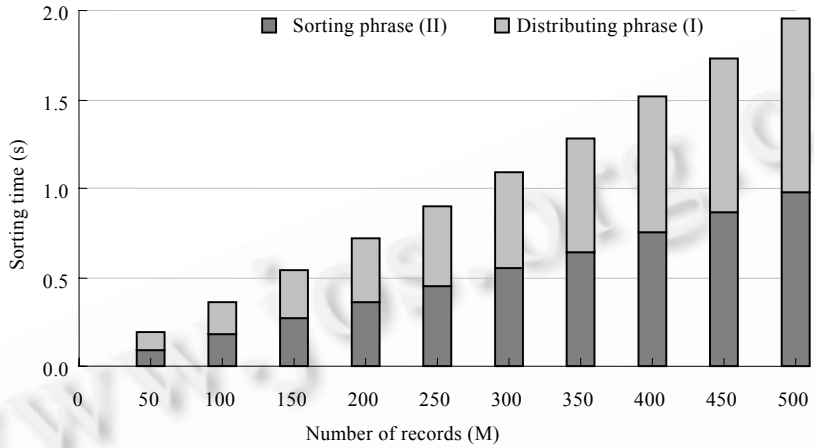

Fig.5 Performance of SheenkSort

图 5 SheenkSort 系统性能

Table 1 Hardware configurations of SheenkSort v.s. THSort

表 1 SheenkSort 及 THSort 排序系统硬件配置

	SheenkSort	THSort
Mainboard	MSI K7N2-L (nForce-2 chipset)	Abit KR7A-RAID (with HPT 372)
CPU	AMD Athlon XP1700+	AMD Athelon XP 1700+
Hard disk	Maxtor 40GB HDD RPM7200×4	IBM 40G HDD ATA 100 IDE 7200RPM×4
Memory	Kingston PC2100 512M DDR SDRAM×2	Kingmax PC2100 512M DDR SDRAM×2

注意到两个系统的排序步骤在实现上非常相似,仅使用的内排序算法不一样.另外,SheenkSort 分发步骤中的读线程与 THSort 的归并步骤的读进程所完成的任务也几乎相同,主要区别在于前者的写线程使用二分查找算法对记录进行分发,而后者则通过堆排序算法对记录进行归并.另外,SheenkSort 使用了前 70K 记录估算子文件边界,但这仅是一个内排序过程,占用时间不超过 1s.

在两个系统的对比实验中,为了排除硬件性能的影响,我们将 THSort 程序移植到 SheenkSort 平台,并在不同数据量下进行对比测试,结果如图 6 所示.受实验条件限制,我们无法给出其在 40GB 及更大数据量下的结果.可以看到,THSort 所使用多路归并算法的排序阶段与归并阶段所消耗时间接近,而总体性能在 SheenkSort 硬件平台上并没有得到明显提高.由于 SheenkSort 在很大程度上得益于其改进的内排序算法,而我们无法修改 THSort,因此,在进一步的对比实验中,将 SheenkSort 系统的内排序模块替换成 Linux 系统下 glibc 中所实现的 qsort 算法,重新实验的结果如图 7 所示.

对比图 7 和图 6 我们发现,两个系统在排序阶段所耗时间接近,因为二者都是对各子文件进行内排序,并且所使用的内排序算法以及系统架构都非常类似.不过,SheenkSort 并非针对快排算法设计(这里也没有对系统参数重新进行调整),而且所使用的 qsort 算法未经优化,排序阶段耗时多出 30%.但 SheenkSort 分发阶段耗时明显少于 THSort 归并阶段,仅为其 1/3,使前者综合性能大约超出后者 17%.对于 PennySort 记录,绝大多数情况下仅比较前 4 个字节就可确定两记录的相对大小,但交换操作却必须处理全部 100 字节,CPU 开销大大超过记录比较.前面分析抽样分点策略处理每条记录约需 $\log_2 M$ 次比较操作,而堆归并算法则需 $2\log_2 M$ 次比较和 $\log_2 M$ 次交换,二者运算开销相差数十倍.文献[10]提到 CPU 已成为 THSort 系统的瓶颈,而 SheenkSort 平台却仅改善了系

统 I/O 性能,这是 THSort 移植后性能没有明显提高,但 CPU 开销较小的 SheenkSort 却能在分发阶段实现高效的原因;也是将 SheenkSort 内排序算法替换成快速排序算法后排序阶段性能降低的原因:针对 PennySort 记录的独立测试表明,SheenkSort 内排序算法效率大约是 qsort 的 5 倍.同时可推知,在 SheenkSort 硬件平台上,如果仅改进 THSort 的内排序算法,其排序阶段时间可以降低到原来的 1/3(参考图 5 中 SheenkSort 系统的排序阶段耗时),但归并时间不会减少,系统总体性能大约提高一半,达到约 14MB/s,仍低于使用桶排序算法的 SheenkSort 系统 28MB/s.

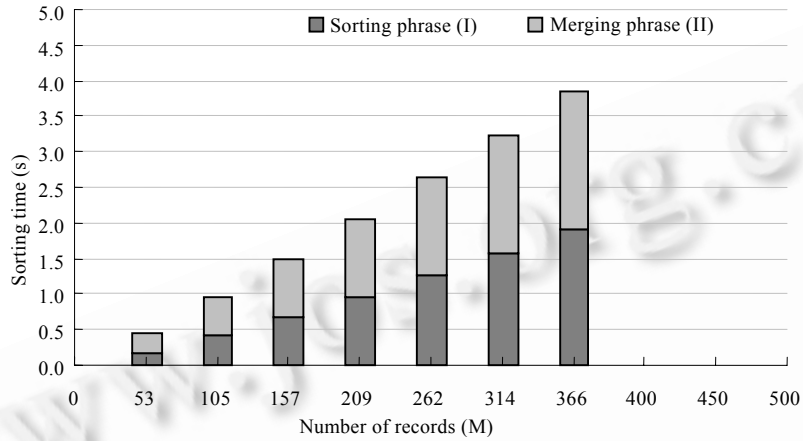

Fig.6 Performance of THSort on SheenkSort platform

图 6 THSort 在 SheenkSort 平台上的排序性能

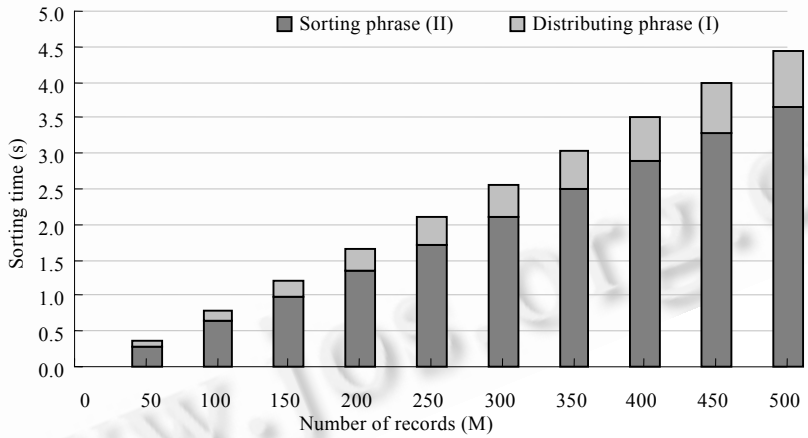

Fig.7 Performance of SheenkSort using qsort

图 7 使用 qsort 内排序算法时 SheenkSort 系统的排序性能

5 结 语

相对于目前常用的多路归并外排序算法,桶排序算法因分发阶段运算简单而能够达到较高的排序效率.但常用的高位分发策略在应用中受到诸多限制而往往无法奏效:关键字非整型,导致无法仅根据高位比特比较记录大小;关键字分布不均匀,导致所得子文件大小不一,内存浪费严重;子文件数无法自由选择,算法很难获得最佳效率.

抽样分点分发策略克服上述困难:用少数抽样记录的 M 均匀等分点作为子文件边界,并通过二分查找算法确定任一记录所属的子文件编号.该策略不再依赖于关键字的具体形式,可应用于整型以外的数据,同时也可以在一定范围内随意调整子文件数.本文粗略分析了抽样分点估计的理论基础,并给出了在一定利用率下不发生

内存溢出的保证概率.上述分析表明,通过抽样分点策略,我们能够得到大小相对一致的子文件.从而,随机抽样分点策略使得桶排序算法能够得到更加广泛的应用.

致谢 我们对于 Jim Gray 先生为 PennySort 比赛所做出的工作表示钦佩.同时也感谢 2002 年度 PennySort 比赛 Daytona 组冠军刘鹏等同学向我们提供的无私帮助.

References:

[1] Owen A. Bubble sort: An archaeological algorithmic analysis. In: Grissom S, Knox D, Joyce D, Dann W, eds. Proc. of the 34th SIGCSE Technical Symp. on Computer Science Education. New York: ACM Press, 2003. 1−5.

[2] Hore CAR. Quicksort. The Computer Journal, 1962,5(1):10−16.

[3] Tang XY. Fast sorting method of separating segment. Journal of Software, 1993,4(2):53−57 (in Chinese with English abstract).

[4] Wang XY. A new sorting method by base distribution and linking. Chinese Journal of Computers, 2002,23(7):774−777(in Chinese with English abstract).

[5] Chen JC. Proportion extend sort. SIAM Journal on Computing, 2001,(31)1:323−330.

[6] Lorin H. Sorting and Sort Systems. Reading: Addison-Wesley Publishing Company, 1975.

[7] Knuth DE. The Art of Computer Programming, Vol 3: Sorting and Searching. Reading: Addison-Wesley Publishing Company, 1973.

[8] Gray J, Coates J, Nyberg C. Performance/Price sort and PennySort. Technical Report, MS-TR-98-45, Microsoft Research, 1998.

[9] Sort Benchmark. http://research.microsoft.com/barc/SortBenchmark/

[10] Shi Y, Zhang L, Liu P. THSORT: A single-processor parallel sorting algorithm. Journal of Software, 2003,14(2):159−165 (in Chinese with English abstract). http//www.jos.org.cn/1000-9825/15/159.htm

[11] Wegner LM, Teuhola JI. The external heapsort. IEEE Trans. on Software Engineering, 1989,15(7):917−925.

附中文参考文献:

[3] 唐向阳.分段快速排序算法.软件学报,1993,4(2):53−57.

[4] 王向阳.任意分布数据的基数分配链接排序算法.计算机学报,2000,23(7):774−777.

[10] 施遥,张力,刘鹏.THSORT:单机并行排序算法.软件学报,2003,14(2):159−165. http//www.jos.org.cn/1000-9825/15/159.htm