实时历史数据库中压缩技术的并行化研究

祝 君[1,2],林庆农[2],徐造林[1]
(1. 东南大学 计算机学院,江苏 南京 210096;
2. 南京南瑞继保电气有限公司,江苏 南京 211100)

摘 要:实时历史数据库系统需要极高的数据压缩速度,但传统的压缩算法都使用串行处理,数据压缩和文件读写都要耗费大量的时间。为了提高数据压缩速度,提出一种并行压缩算法,首先把待处理数据分成若干小块,然后通过对块数据压缩、解压缩、文件读写并发处理,充分发挥多核处理器的高性能。并设计出一种适合并行压缩算法的压缩数据存储文件格式和一个数据段信息循环链表,通过多线程编程技术来实现并行压缩算法。使用工程实测数据在多核处理器系统上的实验表明,并行压缩算法可以极大地提高数据压缩速度,满足实时历史数据库的需求。

关键词:实时历史数据库;数据压缩;并行编程
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2010)07-0036-04

Research on Parallel Compression Technology in Real-Time Historical Database

ZHU Jun[1,2], LIN Qing-nong[2], XU Zao-lin[1]
(1. Computer College, Southeast University, Nanjing 210096, China;
2. Nanjing Nari-Relays Electric Co. Ltd., Nanjing 211100, China)

Abstract: Real-time historical database system needs a high data compression speed. The traditional compression algorithms with serial processing technique require large amount of time to compress data and read or write file. In order to improve data compression speed, a parallel compression algorithm is presented. The high performance of multi-core processors can be fully exploited through the concurrent processing of data compression, data decompression, reading and writing of data files. A file format and a circular linked list for data segment information are designed to fit the proposed parallel compression algorithm. Multi-thread programming techniques are employed to achieve the parallel compression algorithm. Experiment is conducted on a real engineering data with a multi-core processor system. The results show that data compression speed can be greatly improved with the parallel compression algorithm and the requirement of real-time historical database can be met.

Key words: real-time historical database; data compression; parallel programming

0 引 言

近年来,计算机技术、网络技术及控制技术飞速发展,各种实时控制系统在自动化生产中得到了广泛的应用,这些系统在实时生产中需要快速采集大量毫秒级的运行数据,来监测或控制系统的正常运行,并需要把这些海量的实时数据保存起来,供以后分析使用。它们是企业生产状况的具体体现,同时也是分析、优化和指导运行和管理的基础。只有利用实时历史数据库系统对这些实时、历史数据进行自动采集、存储、浏览、分析和研究,才能及时发现目前企业生产中所存在的问题,不断优化企业的生产过程,实现企业的效益最大化[1]。实时历史数据库系统要提供高速的数据采集和数据处理。为了使实时历史数据库系统快速、有效地管理数据,提高磁盘存储效率,就需要有快速的、高精度的数据压缩和解压缩技术的支持。

传统的系统对数据的压缩和对文件的读写都是串行处理,而现在处理器频率的提升已碰到瓶颈,硬盘读写速度也不可能在短时间内有较大的提高,成熟压缩算法的时间复杂度也很难在短时间通过改进得以较大的降低,这就使得要较大地提高数据压缩速度极为困难。但如果用并行的思维去思考这些问题时却会发现还有很大的提高空间,首先是可以对多段数据并行压缩,其次还可以对压缩计算和硬盘读写并行处理,这都

收稿日期:2009-11-10;**修回日期**:2010-02-14
作者简介:祝 君(1983-),男,四川广元人,硕士研究生,研究方向为多核结构与并行程序设计;徐造林,硕士,副教授,研究方向为嵌入式系统、计算机应用。

将很大地提高总体处理速度。随着 Intel 和 AMD 多核处理器不断推出,对数据压缩处理进行并行化的条件已经成熟。文献[2~4]都通过并行化提高了数据压缩的速度。

1　压缩算法原理及应用概述

1.1　压缩算法概述

本质上数据压缩是因为数据自身具有冗余性,数据压缩是利用各种算法将数据冗余压缩到最小,并尽可能地减少失真,从而提高传输效率和节约存储空间。数据压缩技术一般分为有损压缩和无损压缩。无损压缩是指重构压缩后数据得到的重构数据与原来数据完全相同。典型的无损压缩算法有 Shannon－Fano 编码、Huffman 编码、游程(Run－length)编码、LZW(Lempel－Ziv－Welch)编码和算术编码等。而有损压缩是重构压缩后数据得到的重构数据与原来数据有所不同,但不影响原始资料表达的信息,而压缩率一般要大的多。

常用的有损压缩算法有:PCM(脉冲编码调制)、预测编码、变换编码、插值和外推法、统计编码、矢量量化和子带编码等[5]。

1.2　压缩算法在实时历史数据库中的应用

PI(Plant Information System)采用的是旋转门压缩专利技术。旋转门(Swing Door Trending,SDT)[6]在实时过程中用得最多,尽管压缩率不如信号变换方法高,但它的突出优点是算法简单,执行速度快。SDT 压缩算法通过减少保留的数据点个数来实现压缩,适合压缩比较平缓的数据,如果遇到抖动比较剧烈的数据,则必须先进行数据滤波,然后再进行压缩[7]。因此这样的压缩过程对数据是有损的,同时使数据整体丢失原始的动态特征,不适合用于精度要求高的场合。

LZW 算法因其实现较为容易,实现过程没有复杂运算,通过改进后[8]压缩与解压缩耗时短,适用于实时性要求高的场合。LZW 是 LZ(Lemple－Ziv)系列算法之一,是基于字典模型的压缩算法,其原理是以字典的索引号代替它所表示的字符串,在压缩编码的过程中自动生成字典,字典不独立存储,在解压过程中,动态形成与编码过程完全一致的解码字典。

LZO(Lempel－Ziv－Oberhumer)是一种高压缩比和解压缩速度极快的无损压缩算法。LZO 有多个版本,不同版本的压缩率和压缩/解压缩速度也不尽相同,可以根据具体需求来选择。LZO 是使用 ANSI C 编写的一套压缩/解压缩库,遵从 GNU General Public License(GPL)发布,参考实现程序是线程安全的。

实时历史数据库对压缩速度和解压缩速度要求特别高,LZO 很好地解决了压缩率和解压缩速度之间的矛盾,所以 LZO 非常适合实时历史数据库对数据的压缩。文中就采用 LZO 为基本压缩算法。

2　并行压缩算法设计

串行压缩对数据依次压缩,再逐一写入硬盘;串行解压缩则依次从硬盘读出数据,再逐一解压缩。在整个过程中,都只有一个线程、一个核在工作。多线程并行化的核心在于对多个代码段或数据块并发执行,这样可以避免处理器空闲。因此,文中提出一种并行化的方法,采用一个读写线程和多个工作线程并发工作。多个工作线程就可以对多个数据块同时压缩,读写线程也和工作线程并发进行,从而极大地提高压缩速度。

进行并行化一般有两种方法:一是对算法结构并行化,二是对处理的数据并行化。经过分析,LZ 类算法中可并行化处理的计算很少,不太可能通过对算法结构并行化提高性能。但很明显可以对数据并行化处理,首先把数据分为多段,各段之间相对独立,对各段分别进行压缩处理;然后再将压缩后的数据依次写入硬盘。这样可以很大地提高处理的速度,且所需的工作内存也会减少。当然分段压缩是要付出压缩比有所降低的代价,但对于实时历史数据库这种对速度要求明显高于对压缩比要求的环境来讲,牺牲较小的压缩比换取较大的速度提高是值得的、有意义的。

2.1　并行压缩文件格式

并行压缩的文件格式不能使用串行压缩所使用的文件格式,文中设计了一种适合并行化压缩算法的文件格式,具体格式如下:

{原始数据每段的长度 LEN,组成本文件的总段数 N,第 1 段的长度 len_1,第 1 段的数据 data_1,第 2 段的长度 len_2,第 2 段的数据 data_2,…,第 i 段的长度 len_i,第 i 段的数据 data_i,…,第 N 段的长度 len_N,第 N 段的数据 data_N,文件结束符 EOF}

这种文件格式不仅方便压缩时的写入和解压缩时的读出,还可以方便读取其中的一部分数据。传统的文件无法只解压缩其中的一部分数据,必须全部解压缩到内存再取需要的部分,而用新的文件格式可以只读取需要的部分,然后解压缩。

2.2　压缩和写入文件的并行化算法

建立 N 个数据缓存区用于存放压缩后的数据;多个压缩线程并行地压缩原始数据,然后将压缩后的数据写入缓存区,当无空闲缓存区可用时则放弃 CPU,等待有空闲缓存区可用后再继续运行;一个写文件线程将压缩后的数据从缓存区写到硬盘文件中去,写文件完成后将缓存区交由压缩线程重复使用,当无已压

缩好的数据时则放弃 CPU,等待有压缩好的数据后再继续运行。多个压缩线程和一个写文件线程并发执行,这样使得所有硬核同时工作,最大限度地发挥多核处理器的性能,提高压缩速度。

2.3 读取文件和解压缩的并行化算法

建立 N 个数据缓存区用于存放待解压缩的数据;一个读文件线程将待解压缩的数据从硬盘文件读取到缓存区,当无空闲的缓冲区时则放弃 CPU,等待有空闲的缓冲区可用后再继续运行;多个解压缩线程并行地对压缩数据进行解压缩,解压缩完成后将缓存区作为空闲缓存区交给读文件线程重复使用,当无已放好待解压缩数据的缓存区时则放弃 CPU,等待有已放好待解压缩数据的缓存区后再继续运行;多个压缩线程和一个读文件线程并发执行。

3 并行压缩算法实现

多个并发线程之间的工作机制与“生产者/消费者”模型类似:压缩时,工作线程对数据进行压缩并写入缓冲区,完成后由写线程将压缩数据输出到指定文件;解压缩时,读线程首先将数据读到缓冲区,完成后工作线程从缓冲区中取数据并进行解压缩。压缩线程和写线程之间,读线程和解压缩线程之间,后面的线程要用到前面线程所产生的数据,只有前面线程产生数据之后,后面线程才可以使用。同时还要保证写入文件的段顺序要和原始数据的顺序一致,这样在解压缩时才能正确还原数据。

3.1 数据信息循环链表

首先定义段数据信息的结构体:

```
typedef struct
{ unsigned char * psrc; //原始数据的指针
    unsigned long len_src;//原始数据的长度
    unsigned char * plzo; //压缩数据的指针
    unsigned long len_lzo;//压缩数据的长度
    BOOL flag; //本段数据信息已填好标志
    BLOCK * next; //下一段信息的指针
}BLOCK;
```

对这种“生产者/消费者”模型,常用队列来处理数据信息,队列是一种满足先进先出(FIFO)的线性表。对于单线程编程方式可以很好地完成。但在使用并行编程后,多个压缩线程同时进行压缩处理,完成后放入待写队列,由于压缩完成时间不确定,可能后面的数据先完成压缩先进入队列,这样队列就不能保证写文件线程可以按原始数据的顺序将压缩后的数据写入文件。

为了高效准确的并行压缩,文中设计了一种适合并行化编程处理的循环链表。首先生成一个含有 N_token+2 个 BLOCK 的循环链表(N_token,缓存区的个数)。pfile 为写(读)文件所用的指针,pcomp 为压缩(解压缩)所用的指针,pidle 为空闲缓存区指针。为了防止线程之间发生冲突,在这个循环链表设置了锁信号,所有线程利用锁信号进行互斥访问。因为在循环链表中 pfile,pcomp,pidle 都是依次向后移动,压缩数据顺序和原始数据的顺序一致,这样多个压缩线程可以对多个数据段同时压缩,完成后去压缩最靠前的待压缩数据,而写文件线程会严格按原始数据的顺序将压缩数据写入文件。

图 1 为循环链表示意图。

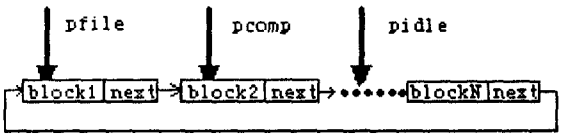

图 1 循环链表示意图

3.2 压缩和写入文件并行化算法的伪代码

① 写文件函数 WriteFileFunc():

```
while(1){
//len_lzo==0 则压缩未完成,让出 CPU 等待
while(pfile->len_lzo==0) Yield();
//将压缩好的数据写入文件
//将刚用完的缓存区交由压缩线程重新使用
if 待压缩数据已全部放入缓存区
pidle->psrc = NULL; //结束标志
else pidle->psrc = ++pdata;//新数据地址
pidle->flag = 1; //数据已准备好
pidle = pidle->next; //pidle 向后移动
pfile = pfile->next; //pfile 向后移动
//psrc==NULL 则所有数据都已压缩并写入文件
if(pfile->psrc==NULL) break;
}
```

②压缩函数 CompFunc():

```
while(1){
//flag==0 则数据未准备好,让出 CPU 等待
while(pcomp->flag==0) Yield();
//对数据进行压缩计算
pcomp->len_lzo = len_lzo; //填入 len_lzo
pcomp = pcomp->next;// pcomp 向后移动
//如果 psrc==NULL 则所有数据都已压缩
if(pcomp->psrc==NULL) break;
}
```

③并行压缩的总体伪代码:

```
{ //首先进行初始化
//然后开启与硬件核数相同数目的压缩线程
for(i=0;i<N_thread;i++)
CreateThread(NULL,0,CompFunc,NULL,0,NULL);
```

```
WriteFileFunc();//主线程作为写文件线程
}
```

多个压缩线程和一个写文件线程并发运行,让所有硬核同时工作,充分利用多核资源来提高压缩速度。

3.3 读文件和解压缩并行化算法的伪代码

解压缩时读文件函数是"生产者",而解压缩函数是"消费者"。参照并行压缩的伪代码很方便写出并行解压缩过程的伪代码。

4 测试数据和分析

实验采用南京南瑞继保电气有限公司的 RCS992 在实际电网中录波数据,数据文件大小为 100MB。分别进行压缩并写入硬盘文件测试、从硬盘文件读出并解压缩测试。测试平台为(CPU:AMD Athlon 64 X4 2.60GHz;内存:1GB;硬盘:7200r/s)。多次测试,得到的平均数据如表 1 所示。

表 1　测试结果

使用的核数	压缩率	压缩(ms)	压缩加速比	解压缩(ms)	解压缩加速比
串行(1)	21.4%	4516	1	234	1
并行(2)	21.5%	2328	1.9	141	1.65
并行(3)	21.5%	1765	2.56	140	1.67
并行(4)	21.5%	1391	3.25	140	1.67

从表 1 数据可知,采用并行压缩算法后压缩率仅升高 0.1%,完全在可接受的范围之内。并行压缩算法以压缩率极小的损失换取了压缩速度和解压缩速度极大的提高。

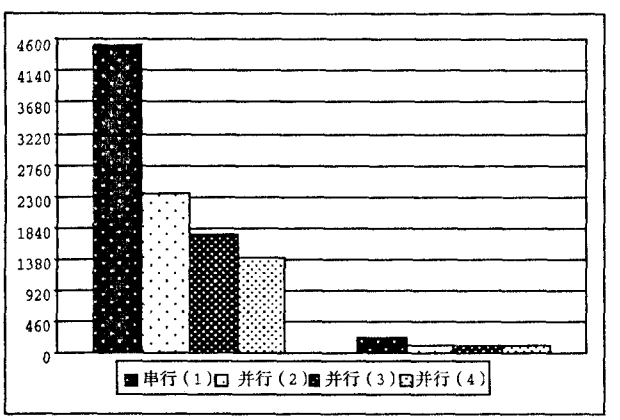

图 2　时间对比图

由图 2 可知,并行压缩算法的压缩速度和解压缩速度都没能同 CPU 核数的增加而得到线性增加,限制加速比的瓶颈在文件读写部分。因为硬盘文件的读写都只能串行操作,而这部分时间无法通过并行化来减少,因此并行压缩算法也不可能得到线性的加速比。但随着硬盘读写速度的提高,并行压缩算法的压缩和解压缩速度也将会跟着提高。

5 结束语

文中提出一种对压缩算法进行并行处理的思想,并进行了编码实现,在多核处理器上的实验表明压缩速度和解压缩速度都得到了很大的提高,满足了实时历史数据库对数据压缩速度的需求。文中只是介绍了从内存到硬盘文件的压缩和解压缩算法,可以按此算法设计出对通过网络传输的数据进行压缩和解压缩的并行算法;进一步可以设计出文件到文件压缩和解压缩的并行算法,通过网络传输文件的压缩和解压缩并行算法。

参考文献:

[1] 拓广忠,慕　群. 实时数据库原理及其压缩技术分析[J]. 华北电力技术,2004(6):17-20.

[2] 宋　刚,蒋孟奇,张云泉,等. 共享存储和 Gzip 的并行压缩算法研究[J]. 计算机工程与设计,2009,30(4):781-784.

[3] Gilchrist J. Parallel compression with BZIP2[C]// Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS2004). MIT, Cambridge, USA:ACTA Press,2004:559-564.

[4] Stauffer L M, Hirschberg D S. Dictionary compression the PRAM [J]. Parallel Processing Letters, 1997, 7(3):297-308.

[5] 李雷定,马铁华,尤文斌. 常用数据无损压缩算法分析[J]. 电子设计工程, 2009(1):49-50.

[6] Bristol E H. Swing Door Trending:Adaptive Trending Recording[J]. Instrument Society of America: Research Triangle Park, 1990:749-754.

[7] 周学文,汤同奎,邵惠鹤. SDT 算法及其在过程数据压缩中的应用[J]. 计算机应用与软件, 2003(1):47-49.

[8] 许　霞,马光思,鱼　涛. LZW 无损压缩算法的研究与改进[J]. 计算机技术与发展,2009,19(4):125-127.