大数据下基于多 CPU 的两级指纹流水计算去重方法①

贺建英, 袁小艳, 唐青松

(四川文理学院 计算机学院, 达州 635000)

摘 要: 分析数据去重的重要意义, 根据现有的数据去重技术和算法, 改进 MD5 码指纹的计算算法并进行优化, 分析并重组指纹计算的流水化方法, 利用缓存组代替单个缓存的方式, 提出一种基于多 CPU 的两级指纹流水计算方法, 对该方法进行分析研究, 并通过相关试验和试验数据来支持该方法的有效性.

关键词: 数据去重; 多 CPU; 两级指纹; 算法; 缓存组

Duplicate Removal Method of Large Data under Two Level Fingerprins Flow Based on Multi CPU Calculation

HE Jian-Ying, YUAN Xiao-Yan, TANG Qing-Song

(College of Computer, Sichuan University of Arts and Science, Dazhou 635000, China)

Abstract: This paper analyzes the importance of data removal. According to the existing data of the removal techniques and algorithms, it improves MD5 code fingerprint algorithm to calculate and optimize it, analyzes and recombines the fingerprint calculation of water level. Using the cache group to replace the single cache, we propose a new method to calculate the two water level fingerprints based on multi CPU to study and analyze the method. At the same time, it supports effectiveness of the method through the relevant tests and test data.

Key words: data duplicate removal; multi CPU; the two level of fingerprint; algorithm; cache group

信息技术的不断进步, 已经使人类社会进入了大数据时代(Big Data Era)[1]. Jim Gray 在 98 年的演讲中曾指出: 由“摩尔定律”提出新的经验定律“未来的 18 个月全球新增存储容量是有史以来全部存储容量的总和”[2]. 在 2010 年, IDC 数据显示, 2010 年全球的信息总量已经达到 1.2ZB, 即 1.2*1021GB, 估计在 2020 年全球的信息总量将达到 35ZB. 同年, Intel 的万亿级项目总监 Jim Held 表示: “全球的海量增长已经达到当前的存储极限, 且每年以 60%的速度递增, 已使得我们无法处理”. 按照上述的两种言论计算, 到 2020 年全球的信息总量将远远超过预期值. 信息量的不断增大 ,数据的存储对现有设备和技术而言提出了新的挑战: 一方面要不断更新技术, 研发能存储大量数据信息的超大容量存储器; 另一方面要考虑到设备的可靠性、扩展性、安全性及吞吐性能. 但如果只是一味地从硬件层面上来解决这些问题, 而不去考虑在存储过程及访问过程中的优化技术, 那么将很难改善在大数据环境下信息存储和访问的瓶颈问题. 在存储过程中可以考虑对现有信息的压缩技术和相同文件内容的去重技术来解决在存储容量和存储空间上的问题. 现有的去重技术已经基本形成, 但从效率上来看, 高低不一, 有的去重方法是以牺牲时间为代价进行去重操作, 而有的去重方法是以牺牲空间为代价进行去重操作. 鉴于目前的现状本文从实际研究出发, 主要针对文件内容的去重研究, 通过对现有的传统去重算法的分析, 得出对文件级数据和块级数据去重方法的基本框架, 分析和改进两级指纹流水计算方法, 并把该方法应用到多 CPU 中, 从而从时间和空间上提高去重效率.

① 基金项目:国家档案局项目(2014-X-65);四川省教育厅一般项目(14ZB0313)
收稿时间:2015-01-28;收到修改稿时间:2015-03-18

1 数据去重的研究现状

数据去重(Data Deduplication)是指对数据集中相同数据或者相似数据的去重操作, 目的是为了减少数据的存储空间, 提高数据的传输率. 目前已有大量的业内研究人员和工业界的相关人士在关注数据去重技术, 也有丰硕的研究成果, 已有相关的算法被设计出来. 在现有的算法中主要有: 只存储相关文件一次的SIS(Single Instance Storage)算法[3], 该算法在 Windows 服务器上经测试可以节省 58%的存储空间; Bell 实验室的 Venti 归档文件系统[4]; LBFS[5]使用 Rabin[6]指纹对长文件内容操作; NFC 公司提出的双峰算法[7]; HP 实验室提出的 TTTD(Two Thresholds Two Divisors)算法[8]; NEC 美国实验室提出的 Fingerdiff[9]新切片算法; Tridegll 和 Mackerras 设计的 Rsync[10]算法; 滑动窗口算法等; 另外还有一些复杂的存储系统如: REBL、Deepstore、DEBAR 等.

数据去重可在源端(客户端)去重, 也可以在宿端(服务器)去重. 在源端去重就是指在客户端准备要传输数据的指纹与服务器端已有数据指纹进行对比, 找到并删除相同数据, 再把非重复数据发送到服务器端, 节省网络带宽的同时又提高了存储的资源. 宿端去重是指把存储文件直接传输给服务器, 并在服务器的内部找到并删除重复内容, 从而减少数据的存储空间. 从现有技术而言主要是从构造内存快速索引、利用缓存、利用数据的相似性和使用一些新型的存储介质等相关技术来解决问题.

2 多CPU并行的两级指纹流水方法

2.1 基本框架

文件数据的去重可以在文件级、块级和字节级中进行. 文件级的去重是把文件作为一个整体来计算其 MD5 码值判断是否重复, 对于去重的目标是一个自然文件, 这个方式需要的消耗是最小的, 其缺点就是假设只是对文件做了一个很小的修改, 即便是一个标点符号, 也需要对该文件重新计算其 MD5 码值, 形成了不同的两个文件, 这样前面节省的空间就不覆存在, 这种去重方式适合于较小的文件. 对于块级去重可以分为定长和可变长的块级去重, 相比文件级的去重, 需要更多的消耗, 但对于大文件来说是非常有用的, 因为对于大文件来说重复数据的机率是较大的. 在对字节级去重时, 需要更多的消耗去确定重复数据的开始和结束区域. 根据他们的特点前辈们将文件级数据和块级内容去重相结合的方式进行研究, 在这种方式中首先判断文件级的块, 如果文件已经被识别为重复数据, 则不需要再对文件内容进行分块进行下一步去重. 其具体框架如图 1 所示.

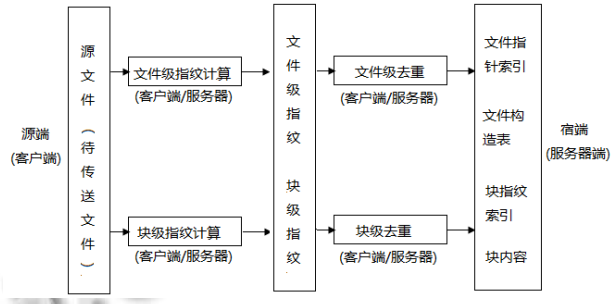

图 1 去重基本框架

如图 1 所示, 把源端(客户端)的文件上传到宿端(服务器)中, 在这个过程中文件级的指纹计算和块级的指纹计算既可以放在客户端进行也可以放在服务器端执行, 根据得到的指纹既可以在客户端实现去重操作, 也可以在服务器端实现去重操作. 这样如果判断文件为重复文件就直接在客户端或者服务器端去掉该文件, 如果不是重复文件则进一步划分为不定长的分块序列进行块级去重. 对发生在客户端的去重可以节省网络带宽和存储资源, 只需要发送非重复的数据到服务器端. 对发生在服务器端的去重则是实现在线去重, 对网络的要求较高. 计算指纹若在客户端, 则首先读取文件内容并计算其文件 MD5 码值, 再读取文件内容, 进而计算其分块算法并计算指纹序列. 设读取存储器中源文件的吞吐率为 FC_r, 读取分块的吞吐率为 FC_k, 读取 MD5 码值的吞吐率为 FC_h, 文件的长度为 L, 文件级指纹的耗时为 $Time_f$, 块级指纹的耗时为 $Time_c$, 发生在客户端的时间开销为 $Time_{client}$, 其表示结果如式子(1)所示:

$$Time_{client}=Time_{file}+Time_{chock}$$
$$=(L/FC_r+L/FC_h)+(L/FC_r+L/FC_k+L/FC_h) \quad (1)$$

如果信息被存储在内存中, 则需要的时间要少, 因其减少了在从硬盘读数据到内存的过程.

假如在服务器端计算指纹, 则需要通过网络接受数据, 此时还要考虑到网络的吞吐率, 所计算两级指纹的时间将增加一个 L/FC_i, 其中 FC_i 是网络的吞吐率, 如式子(2)所示. 同理, 如果信息是存在在内存中, 则读取数据的时间开支也相应会减少.

$$Time_{server}=Time_{file}+Time_{chock}$$

$$=(L/FC_r+L/FC_t+L/FC_h)+(L/FC_r+L/FC_t+L/FC_h) \quad (2)$$

2.2 两级指纹算法及优化

所谓两级指纹是指文件级的指纹和块级的指纹，通过分别计算文件级的数据指纹和块级的数据指纹，利用 2.1 节所提到的两级指纹的去重框架，要么在源端去重，要么在宿端去重. 在计算两级指纹时可以利用哈希值来计算其指纹，如文献[11]中提到的两级哈希值并行计算方式，而在本文中则是利用MD5码值计算两级指纹，并通过改进计算MD5码值得算法来提高指纹计算的效率.

在两级去重中使用 MD5 算法[12]计算文件指纹着手，MD5 算法是先对要处理的原始数据进行预处理操作，把原始数据分成 512bit 的整数倍后，再进行报文的分组划分和逻辑处理[13]. 但在实际操作中，并不是所有传入的初始值都会是 512 的整数倍，故本算法在文献[11]的基础上又进行了改进，对划分的值 512bit 稍作修改，转换成 2^n，即 256bit、128bit 来预处理原始数据. 其流程图如图 2 所示.

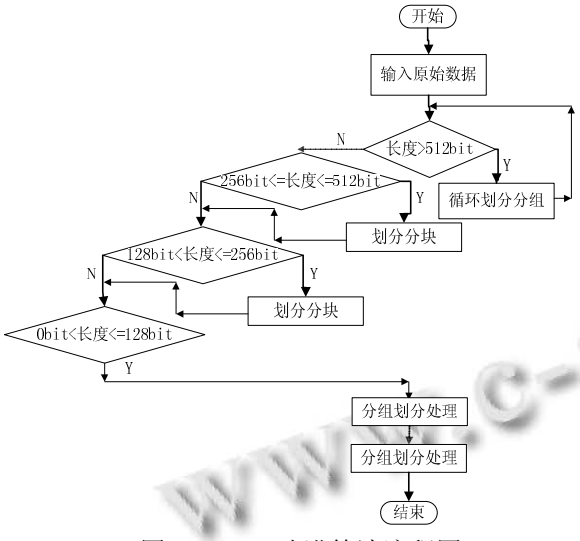

图 2　MD5 改进算法流程图

在图 2 所示的分组方法中，根据长度的不同将按照预设的几个条件来判断其处理过程:

1) 对原始数据长度(*Length*)小于等于 512bit 判断条件为:

a. 256bit<*Length*<=512bit

b. 128bit<Length<=256bit

c. 0bit<Length<=128bit

2) 对原始数据长度(*Length*)大于 512bit 的，则继续循环对初始数据按照 512bit 进行划分，划分完后，利用 1)中的条件继续对分组的数据进行划分.

3) 通过 1)和 2)两个步骤，得到的是 128bit 的消息信息输出.

通过以上的方式分组判断，利用循环进行相应的重复操作，可以节省系统资源，降低运算强度. 这种算法不仅适合计算文件级的指纹，也适合块级指纹的计算，在去重研究中非常有效. 其伪代码如下所示.

先定义一个名为 MD5DataType 的数据类型:

```
MD5DataType
{
char   MD5StringValue[]   //存放四个链接的数组;
int    MD5Length[]   //存放数据的长度，为 64 位;
int    MD5Buffer512[]    //存放当前等待处理的报//文分组，长度为 512 位;
int    MD5Buffer256[]    //存放当前等待处理的报//文分组，长度为 256 位;
int    MD5Buffer128[]    //存放大小为 128 位的报//文分组内容;
}
```

定义方法 MD5Action()来根据长度确定分组划分和逻辑处理:

```
MD5Action(String allmessage, MD5DataType message){
 如果 allmessage.length<=512/8
 则判断 128/8<=allmessage.length<=256/8 条件成立
 则 Copy(message.buffer[],allmessae);//把消息拷贝到分组缓存中;
 且把报文进行逻辑分组处理;
 否则当数据的长度超过 512/8 位时，则对数据报文仅需进行分组处理;
}
```

定义方法 StringToMD5()，用来把原始消息转换成 128bit 的分组;

```
StringToMD5(String allmessage){
 MD5DataType message;
 MD5InitValue(messae);   //定义方法初始化待分组划分的数据;
```

MD5Action(allmessage,message); //定义方法根据长度来确定分组划分和逻辑处理;

}

2.3 以流的方式将各计算任务部署到多个处理器

从两级指纹去重的基本框架 2.1 节的分析可知, 在两级指纹的计算流程中读取文件、接收文件、对文件进行分块、计算数据的 MD5 码消耗了大部分的时间开销, 极易造成速度性能上的瓶颈. 由于这些原因可考虑使用多 CPU 及共享内存并行系统的程序设计等相关技术手段来优化流程. 在该指纹计算流程中不同的任务在执行时以顺序执行为主, 即分块算法的执行必须是在执行文件级 MD5 码指纹之后, 而块级 MD5 码指纹的计算操作也必是在分块算法之后. 在本算法中, 为了防止出现不必要的数据依赖, 将采用缓存组的方式存放相应的数据内容和分块边界, 以便替代单个缓存的形式. 将指纹的计算流程组合成数据源的接收、数据的分块、指纹的计算几个步骤, 并设置在执行时的每一个步骤中, 同一个时刻都将使用一套单个的数据缓存和边界. 其具体执行流程如图 3 所示.

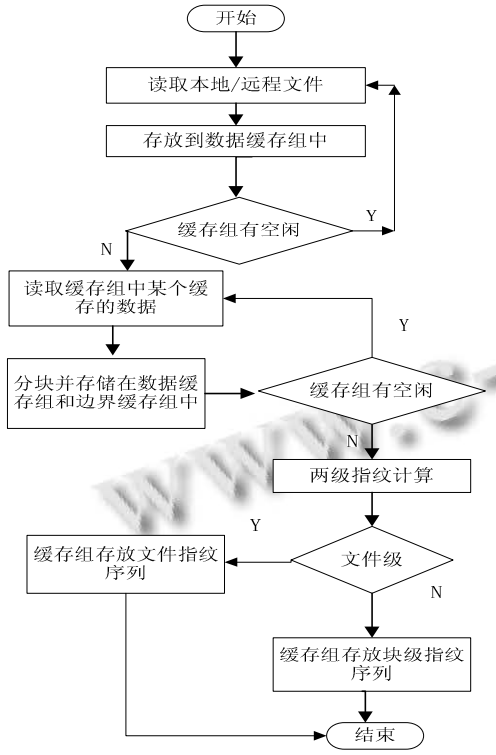

图 3 两级指纹执行流程图

通过流程的执行过程可知, 对数据的输入, 分块操作和两级指纹计算至少需要四个 CPU 来完成流水化操作, 此时在源端计算文件的两级指纹需要的时间开销为:

$$Time_{client} = Max(L/FC_r, L/FCk, L/FC_m) \quad (3)$$

其中 L 为文件的长度, FC_r、FC_k 如前面所述分别为读取文件的吞吐率和分块的吞吐率, 在本算法中采用的是计算文件的 MD5 码指纹, 因此用 FC_m 来表示读取 MD5 码指纹的吞吐率. 此算法将忽略掉入流水线流满等的其他异常情况. 同理, 在宿端计算的时间开销为:

$$Time_{server} = Max(L/FC_r, L/FC_k, L/FC_i, L/FC_m) \quad (4)$$

其中 FC_i 为网络的吞吐率. 从式子(3)和(4)可以看出, 利用多核计算的时间明显低于单核计算的时间.

3 实验结果与分析

为对该算法进行测试, 特采用 Win7 操作系统, MyEclipse10.0 和 JDK1.7, 使用 Java 语言, 编写模拟测试代码, 测试中分别采用单核 Intel Processor 1.73GHZ, 512M 内存; 双核 Intel 2.9GHZ 处理器, 4GB 内存; 及四核 2.5GHZ 处理器, 4GB 内存. 因网络的吞吐率对算法的性能有重要的影响, 而不同带宽对网络的传输率也有影响, 故在测试中为了得到较为稳定的结果, 测试的操作都在客户端进行. 在单核处理器中, 同样把操作分为数据的读取、分块和指纹的计算, 这一系列的任务都在同一个处理器中完成, 故所有的操作都必须排队等待, 其调动过程如图 4 所示.

图 4 单核处理器的调动过程

这种执行过程需要等待一个任务执行完成, 才能执行下一个任务. 在流程中需要不断维护数据缓存和边界缓存, 以便消除缓存的竞争. 在实验中以处理全部数据使用的时间来计算平均指纹计算性能. 测得在 1MB 的读写单元下单核单线程顺序读取的平均吞吐率为 32.4MB/S, 随机读取本地硬盘的平均吞吐率为 28.9MB/S. 在不同的缓冲量下, 本文算法及直接使用线程的处理性能如表 1 所示.

表 1　单核下不同缓冲容量的处理性能

缓冲容量(MB)	1	2	4	8	16	32	64	128	256
平均指纹计算性能(MB/S)	27.4	27.8	29.4	29.8	30.1	30.8	31.2	31.7	32.5
单核单线程(MB/S)	18.8	19.1	19.7	20.1	20.9	21.1	21.7	22.1	22.5
单核双线程(MB/S)	16.5	20.2	21.8	22.7	23.9	25	26.4	27.5	28.7

从表 1 中可以看出，缓冲容量在 2^0MB—2^8MB 之间，本算法的指纹吞吐率在 27.4MB/S—32.5MB/S 之间．相同条件下单核单线程的吞吐率在 16.5MB/S—28.7MB/S 之间．可见即使线程数增加，但其性能的提升也低于本算法的平均指纹计算性能．本算法的优势已有体现．

在双核处理器中，算法的调度过程如图 5 所示．

图 5　双核处理器的调动过程

同理，在不同的缓冲量下，双核处理器对本文算法及直接使用线程的处理性能如表 2 所示．

表 2　双核下不同缓冲容量的处理性能

缓冲容量(MB)	1	2	4	8	16	32	64	128	256
平均指纹计算性能(MB/S)	63.2	63.8	64.2	64.9	65.1	65.7	66.2	67.2	67.8
双核单线程(MB/S)	30.8	31.2	32.7	33.4	34.8	35.3	36.9	38.9	43.7
双核双线程(MB/S)	22.8	27.4	30.8	35.2	37.8	39.7	42.4	47.9	54.8

从表 2 中可以看出，双核中本算法的优势越发明显，在缓冲容量为 256MB 时可以达到 67.8MB/S，而即使是在双核中不同的线程下进行处理，双线程的速度在 256MB 时也只能达到 54.8MB/S.

在四核中，算法的调度过程如图 6 所示．

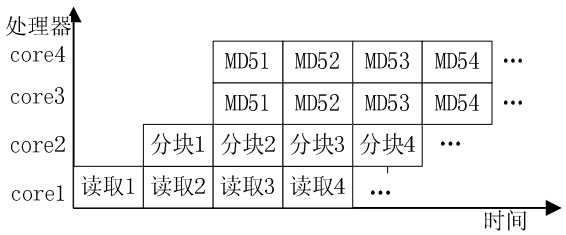

图 6　四核处理器的调动过程

在四核环境下，以流水的方式部署到全部处理器中，使用了三套数据缓存和边界缓存．在不同的缓冲量下，四核处理器对本文算法及直接使用线程的处理性能如表 3 所示．

表 3　四核下不同缓冲容量的处理性能

缓冲容量（MB）	1	2	4	8	16	32	64	128	256
平均指纹计算性能（MB/S）	101.2	102.5	102.9	103.1	103.8	104.1	104.7	104.9	105.3
四核双线程（MB/S）	43.8	44.5	45.2	45.8	46.7	47.9	48.8	50.2	51.7
四核四线程（MB/S）	72.7	73.8	75.2	77.8	79.5	81.7	84.8	85.7	89.1

表 3 可以看出，四核中算法的平均指纹计算性能的优势远比双核要大，可以达到 105.3MB/S，即使使用四核的四线程处理的方式在 256M 的缓冲容量下也只能达到 89.1MB/S，进一步表明在四核中使用本算法可大大加速两级指纹的计算过程．

两级指纹流水加速方法在一次扫描中，能同时生成对应的文件级指纹和相应的块级指纹．但在其他的去重方法中，仅仅是对非重复的数据文件进行分块计算指纹．故从这点上说本算法造成不必要的资源开销．如果要进一步优化该算法，将会从这方面着手考虑．

4　总结

文件级的去重是大粒度的侦测和去重操作，能有效地降低块级去重的计算和查找复杂度，而块级去重则能提高去重的精度，但会消耗更多的时间和空间资源．本文把文件级和块级的去重相结合的方式，并把它们以顺序流水的方式部署在多个处理器中，在算法中采用缓存组的方式消除了除计算以外的数据依赖和缓存竞争．在多核处理器中，减少了重复的硬盘访问所需的时间，进一步提高指纹计算的性能，对重复内容的去重大大的提高了效率，本文的研究对数据文档的去重有一定的实际参考意义．

参考文献

1 Manyika J, Chui M, Brown B, et al. Big Data: The Next Frontier for Innovation, Competition, and Productivity. McKinsey Global Institute, 2011.

2 Gray J. What Next A Dozen Information-Technology Research Goals[Technical Report]. Microsoft Research, 1999. MS-TR-99-50.

3 Bolosky WJ, Corbin S, Goebel D, Douceur JR. Single

instance storage in Windows 2000. Proc. of the 4th USENIX Windows System Symposium, August 2000.

4 Quinlan S, Dorward S. Venti: a new approach to archival storage. Proc. of the First USENIX Conference on File and Storage Technologies. Monterey, CA, USA. 2002.

5 Muthitacharoen A, Chen B, Mazieres D. A low-bandwidth network file system. Proc. of the Symposium on Operating Systems Principles. 2001. 74–187.

6 Rabin MO. Fingerprinting by Random Polynomials[Technical Report]. Center for Research in Computing Technology, HarvardUniversity. May 1981. TR-15·81.

7 Kruus E, Ungureanu C, Dubnicki C. Bimodal content defined chunking for backup streams. Proc. Soft the 8th USENIX Conference on File and Storage Technologies. USENIX Association. 2010.

8 Eshghi K, Lillibridge M, Wilcock L, Belrose G, Hawkes R. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service. Proc. of the 5th USENIX Conference on File and Storage Technologies. 2007.

9 Bobbarjung R, Jagannathan S. Improving duplicate elimination in storage systems. ACM Trans. on Storage, 2006, 2(4): 424–448.

10 Rasch D, Burns R. In-place rsync: File synchronization for mobile and wireless devices. Proc. of the USENIX Annual Technical Conference. June 2003.

11 魏建生.高性能重复数据检测与删除技术研究[学位论文].武汉:华中科技大学,2013.

12 刘俊辉.MD5 消息摘要算法实现及改进.福建电脑,2007,(4):96–97.

13 廖海生.基于重复数据删除技术的数据容灾系统的研究[学位论文].广州:华南理工大学,2009.

14 俞枫,王引娜.基于 DRPKP 算法的文本去重研究与应用.微型电脑应用,2014,(1):58–60.

15 孙有军,张大兴.海量图片文件存储去重技术研究.计算机应用与软件,2014,(4):56–57.

16 成功,李小正,赵全军.一种网络爬虫系统中 URL 去重方法的研究.中国新技术新产品,2014,(12):23.

17 李伟.Web 记录自动抽取与去重方法的研究与实现.西安电子科技大学,2014,(2):25–35.

18 高翔,李兵.中文短文本去重方法研究.计算机工程与应用,2014,(16):196–201.

19 雷德龙,郭殿升,陈崇成,等.基于 MongoDB 的矢量空间数据云存储与处理系统.地理信息科学,2014,(7):508–514.

20 杨祥清.存储系统数据去重策略研究.信息通信,2014,(8):132.