CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 38 卷第 10 期 2016 年 10 月
Vol. 38, No. 10, Oct. 2016

文章编号:1007-130X(2016)10-1967-07

基于位串内容感知的数据分块算法*

周 斌[1],朱容波[1],张 莹[2]

(1.中南民族大学计算机学院,湖北 武汉 430074;2.华中科技大学外国语学院,湖北 武汉 430074)

摘 要:针对基于内容的可变长度的分块 CDC 算法中数字签名计算需要耗费大量 CPU 开销的问题,提出了一种基于位串内容感知的数据块分块算法。算法利用每一次失败匹配尝试所带来的位特征信息,最大限度地排除不能匹配的位置,从而获得最大的跳跃长度,减少中间计算和比较的开销。实验结果表明,本算法减小了数据分块过程中数字签名计算的开销,降低了确定块边界时的 CPU 资源消耗,从而优化了数据分块的时间性能。

关键词:位串内容感知;数据分块;数字签名

中图分类号:TP311.52 **文献标志码**:A

doi:10.3969/j.issn.1007-130X.2016.10.003

A bit string content-aware data chunking algorithm

ZHOU Bin[1], ZHU Rong-bo[1], ZHANG Ying[2]

(1. College of Computer Science, South-Central University For Nationalities, Wuhan 430074;
2. School of Foreign Languages, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: Aiming at the problem of a large amount of overhead introduced by the content defined chunking algorithm (CDC) in calculating the digital signature, we present a novel data chunking algorithm based on bit string content awareness. The proposed algorithm eliminates unmatched positions to the utmost by taking advantage of the bit feature information acquired through each failure matching. Since the maximum jump length is obtained, intermediate calculation and comparison cost are reduced. Experimental results show that the algorithm can reduce the overhead of digital signature calculation in the process of data chunking, cut down CPU resource consumption for chunk boundary determination, and optimize the time performance of data chunking.

Key words: bit string content-aware; data chunking; digital signature

1 引言

相比较固定长度的数据分块(Chunking)算法,You 等[1]已经论证了 CDC(Content Defined Chunking)可以发现更多冗余数据,达到更高去重效率。CDC[2-4]本质实际上仍是一个字符串匹配问题,它将待处理数据对象看成是一个文本(Text),并给出一个长度为 n 的子串作为模式(Pattern),发现数据块(Chunk)边界的过程实际上就是字符串模式匹配的过程。为了简化匹配过程,减少匹配计算量,CDC 并没有直接进行字符串匹配来进行分块,而是通过比较一个预先提出的模式数据指纹和正文中长度为 n^l 的子串数据指纹是否相等来发现 Chunk 边界。这是由于数据指纹的长度相比较原有正文字符串长度要短很多,可以大量地减少字符串匹配计算量。

* 收稿日期:2016-03-18;修回日期:2016-05-03
基金项目:国家自然科学基金(61272497);湖北省自然科学基金(2013CFB447)
通信地址:430074 湖北省武汉市中南民族大学计算机学院
Address: College of Computer Science, South-Central University for Nationalities, Wuhan 430074, Hubei, P. R. China

2 相关工作及问题

现有的数据分块算法大多关注的是分块的不稳定性问题,即在原数据流中插入或者删除少量字节,其冗余数据块的发现率将大大降低。敖莉[5]和孙爱玲等[6]通过采用 Rabin 指纹算法对数据进行分块,虽然解决了不稳定性问题,但其数据块大小相差很大,使得数据块的长度划分成为一个问题。郑亚光等[7]则提出了一种基于回溯匹配的滑动分块 SBBW(Sliding Blocking algorithm with Backtracking Window)算法,对前述算法进行了改进,但实际效果不是太好。

当面临大数据环境时,分块算法不仅需要解决分块的不稳定性问题,更重要的是要解决一个 Chunk 到底划分为多大才能够最大限度地提高冗余数据的发现率,同时又使得数据块元数据开销能够在系统性能承受范围之内的问题。

Muthitacharoen 等[8]在低带宽网络系统 LBFS(Low Bandwidth network File System)的实现中,针对数据分块过程中可能出现的一直不能发现块边界的病态现象,引入 Chunk 长度的上下边界,使得 Chunk 最小不能小于 2 KB,最大不能大于 64 KB,从而避免过多短 Chunk 以及超长 Chunk 两种极端情况的发生。但是,在这种设置下,Chunk 的预期长度和实际划分长度仍然存在较大的误差,有时能达到 50%左右,从而会极大地影响整体性能。

为了更精确地预测 Chunk 的平均长度,韩国汉阳大学的 Jaehong 等[9]提出如下模型:

$$E(S) = S_{\min} + \sum_{i=1}^{N} i * p * (1-p)^{i-1} + N * \left[1 - \sum_{i=1}^{N} p * (1-p)^{i-1}\right] = S_{\min} + 2^n * [1-(1-p)^N] \quad (1)$$

其中,$E(S)$表示 Chunk 的平均长度,$S_{\min}$与 $S_{\max}$分别表示 Chunk 的最小长度和最大长度,并且令 $N=S_{\max}-S_{\min}$,p 表示签名和模式匹配的概率,n 为目标模式中包含的二进制位,$p=(1/2)^n$。实验表明,利用此模型,能使得预测与实际的误差保持在 1%的范围内。

为了使得 Chunk 大小既能取得较好的冗余数据发现率,又能减轻元数据开销对整个系统性能的负面影响,本文借鉴了 Muthitacharoen 与 Jaehong 的研究成果,在测试系统中,设置 Chunk 预期长度为 8 KB。

通过数字签名计算来进行 Chunk 边界划分是一个 CPU 密集型的过程。假设对于一个 100 MB 的文件,如果预期 Chunk 的平均长度为 8 KB,最小长度为 2 KB,那么,文件预期将被划分为 12 800 个 Chunk。由于最小长度为 2 KB,那么前 2 KB 的数据不用进行数字签名运算,对于每个预期 8 KB 的数据块,其 Chunk 边界的获取将要进行8 192−2 048,即 6 144 次签名计算和比较。那么对于整个文件来说,仅仅 100 MB 大小的文件就需要进行 12 800 * 6 144 次签名计算和比较。而在这么繁重的 CPU 运算中,Chunk 的数量与签名计算和比较总次数的比率仅仅为 1∶6 144。无效的签名计算和比较耗费了大量宝贵的 CPU 资源。

3 基于位串内容感知的数据分块算法 BCCA

已有的众多数据块划分[10]的实现中,都采用了 Rabin 算法而不是安全哈希算法 SHA(Secure Hash Algorithm)或者 MD-5(Message Digest algorithm-5)来进行数字签名计算。这是由于 Rabin 算法可以减少很多计算量,但仍然存在很多无效的却是必不可少的签名计算。并且每次进行签名计算,都需要进行求模运算,而这种运算是需要耗费大量 CPU 资源的。

为了减少签名计算消耗大量的 CPU 资源,需要解决的关键问题有两个:第一,尽量减少数据指纹获取的计算量;第二,尽量减少数字签名计算的次数。

BCCA(Bit-string Content-aware Chunking Algorithm)根据二进制位串自身的特点,在大大减轻数据指纹生成开销的同时,将文件数据稳定分块的问题转换为两个二进制位串匹配的过程。并通过每次匹配过程中获得最大的跳跃长度,减少中间的计算和比较开销,从而减小了 Chunking 过程中数字签名计算的开销,提高匹配效率。

其原理如下:

(1) 基于二进制位操作的快速数据指纹生成算法。

Rabin 算法的基本思路就是将一个长度为 m 的模式串 *Pattern* 映射为一个二进制的位串数据(数据指纹),然后将长度为 n 正文 *Text* 的每一个长度为 m 的子串也都映射为二进制的位串数据,通过比较判断两者是否相等。

Rabin 算法中,每次的基本比较单位是字符。为了加快算法的速度,BCCA 考虑仅通过选取每个基本比较单位中的某一个二进制位来代表此单元,这个过程完全可以通过位操作指令完成,而不需要使用求模操作,从而可以大大降低生成数据指纹的 CPU 开销。

因此,BCCA 选择每一个基本比较单元的最低位构成其数据指纹。如果知道了字符串 T_s($T_s=[b_s,b_{s+1},\cdots,b_{s+m-1}]$)的数据指纹为 $f(T_s)$,那么,$f(T_{s+1})$通过移位位操作即可获取。

(2)BCCA 算法的实现。

设原模式子串包含 m 个基本单位,取其每个基本单位的最低二进制位构成 m 位模式数据指纹 *Pattern*;并预取 n 个正文基本单位的最低位存入到位数组 BT 中,根据快速数据指纹生成算法计算正文数据指纹值 $f(T_s)$。

本文考虑采用如下两种方式进行模式匹配:

①基于模式 *Pattern* 的 BCCA BCCA-P(BCCA based on Pattern)。

BCCA-P 采用从右向左的扫描方式进行模式匹配。但是,为了减轻匹配计算量,首先并不是将整个 *Pattern* 进行匹配,而是如图 1 所示,先将 *Pattern* 与 *Text* 进行左对齐,在与 *Pattern* 对齐的 *Text* 尾部选取长度为 $len(x)$的子位串,并从 *Pattern* 右边开始,搜索匹配的子位串。

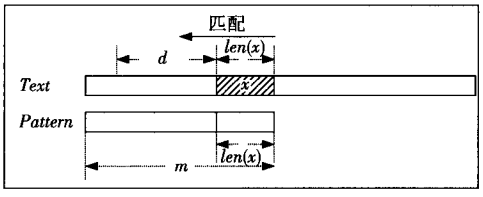

Figure 1　Matching the bit-substring of *text* in *Pattern*

图 1　从 *Pattern* 中搜索 *Text* 匹配子位串

由于比较的是位串,每次比较一个子位串,通过一次位操作就可以完成。文中分两种情况分别考虑:

a 正文位串 *Text* 全匹配。

如图 2 所示,正文位串 *Text* 全匹配是指在匹配过程中,如果每次都能匹配成功,那么 *Text* 中的子位串长度增加 1;然后再与 *Pattern* 中尾部长度相同的子位串进行匹配,直到子位串长度增加到 m。匹配方向为从右向左进行。

如果中间有一次没有匹配成功,那么进行后面的'正文位串 Text 好后缀匹配'操作。

如果$(m-x)+1$ 次匹配都成功,表明此时在 *Pattern* 中找到了与 *Text* 子串匹配的子串,正文

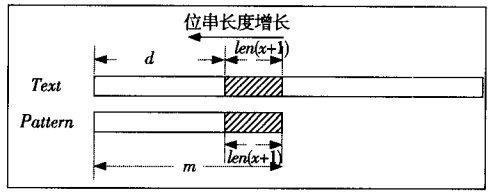

Figure 2　Full matching of *Text*

图 2　正文位串 *Text* 全匹配

位串 *Text* 全匹配成功。

此时需做进一步精确比较,即进行实际字符串比较。如果实际字符串不匹配,则继续进行子串 x 的匹配;否则标记此处为块边界,然后跳过 2 048 位,即 $d=d+2\,048$,继续下一轮比较(规定最小分块长度为 2 048 Bytes)。

b 正文位串 *Text* 好后缀匹配

如图 3 所示,正文位串 *Text* 好后缀匹配是指匹配过程中,*Text* 中长度为 $len(x)$的子位串每次与 *Pattern* 中对应长度的子位串进行匹配。当匹配不成功时,*Pattern* 位串向左滑动一个位置,重新选择长度为 $len(x)$的子串与 *Text* 中的子位串进行比较,直到在 *Pattern* 位串找到匹配子串。这个过程称为正文位串 *Text* 好后缀匹配。

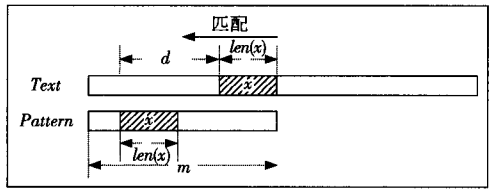

Figure 3　Good suffix matching of *Text*

图 3　正文位串 *Text* 好后缀匹配

当发生了正文位串 *Text* 好后缀匹配时,*Pattern* 整体向右跳跃的距离为 d。并且在匹配过程中,为了尽量减少匹配的计算量,应该尽量获得更大的跳跃距离。但是,在这种方式中,如图 4 所示,最大的跳跃距离为 *Pattern* 的长度,也就是在 *Pattern* 中没有找到子位串匹配,此时 *Pattern* 整体向右跳跃 m 位。

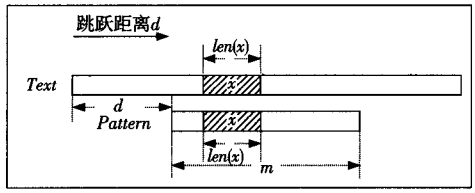

Figure 4　Distance d jumping to right($d\leqslant m$)

图 4　向右跳跃距离 $d(d\leqslant m)$

②基于正文 *Text* 的 BCCA BCCA-T(BCCA based on Text)。

与①相似,BCCA-T 也采用从右向左的扫描方

式进行模式匹配，为了减少匹配的计算量，没有将整个 *Pattern* 进行匹配，而是如图 5 所示，先将 *Pattern* 与 *Text* 进行左对齐；与①不同的是将在 *Pattern* 的最右边开始选取长度为 $len(x)$ 的子位串，在正文位串 *Text* 的 $m-len(x)$ 位置处开始，搜索匹配的子位串。

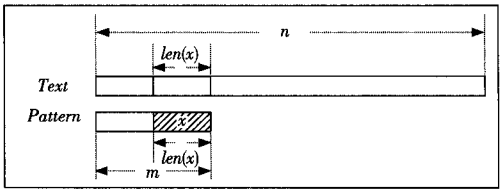

Figure 5 Matching bit-substring in *Text*

图 5 从 *Text* 中搜索匹配子位串

文中也是分以下两种情况分别考虑：

a 模式位串 *Pattern* 全匹配。

如图 6 所示，模式位串 *Pattern* 全匹配是指在匹配过程中，如果每次都能匹配成功，那么 *Pattern* 中的子位串长度增加 1；然后再与 *Text* 中尾部长度相同的子位串进行匹配，直到子位串长度增加到 m。

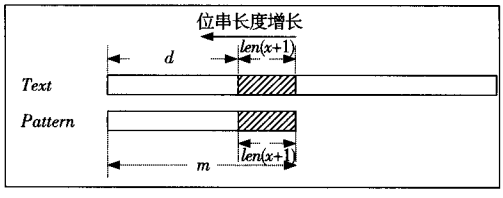

Figure 6 Full matching of *Pattern*

图 6 模式位串 *Pattern* 全匹配

如果中间有一次没有匹配成功，进入模式位串 *Pattern* 好后缀匹配。

如果 $(m-x)+1$ 次匹配都成功，表明此时在 *Text* 中找到了与 *Pattern* 匹配的子串，模式位串 *Pattern* 全匹配成功。

此时也需做进一步的精确比较，即进行实际字符串比较。

b 模式位串 *Pattern* 好后缀匹配。

如图 7 所示，模式位串 *Pattern* 好后缀匹配是指每次匹配的过程中，*Pattern* 中的长度为 $len(x)$ 的子位串每次与 *Text* 中的对应长度的子位串进行匹配。当匹配不成功时，*Text* 位串向右滑动一个位置，重新选择长度为 $len(x)$ 的子串与 *Pattern* 中的子位串进行比较，直到在 *Text* 位串找到匹配子串。这个过程称为模式位串 *Pattern* 好后缀匹配。

当发生了模式位串好后缀匹配时，*Pattern* 整体向右跳跃的距离仍然为 d，但此时由于扫描是从

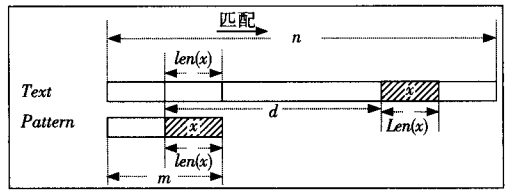

Figure 7 Good suffix matching of *Pattern*

图 7 模式位串 *Pattern* 好后缀匹配

左到右进行的，如图 8 所示，d 的距离可以远远大于 *Pattern* 本身的长度，最大可以到达 $n-len(x)$。

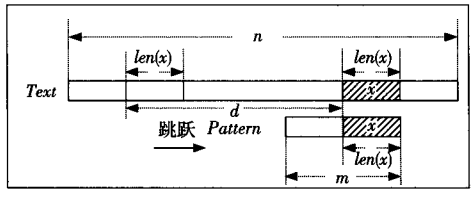

Figure 8 Distance d jumping to right$(1<d<(n-x))$

图 8 向右跳跃距离 $d(1<d<(n-x))$

4 性能评测

本文测试主要分析 BCCA 不同长度的目标模式位串对数据块划分的速度、大小以及压缩率的影响。

4.1 实验环境设置

实验原型系统包括两个节点，每个节点服务器的 CPU 是具有 24 GB DDR RAM 的双路四核至强(2-way quad-core Xeon)E5420，其主频 2 GHz，每个核的缓存为 6 114 KB。Chunk 存储在带有两个磁盘(Seagate Barracuda 7 200 RPM，每个硬盘容量为 2 TB)的阵列 RAID 0 中，每个节点配备一个 Intel 80003ES2LAN gigabit 网卡与 gigabit Ethernet 相连接，其中一个节点作为主服务器，而另一个节点作为镜像服务器。

测试数据集包含了五个子数据集，每个子数据集大小为 10 GB 左右。此处的 iso 文件是选取的一些源代码文件、数据文件等在实际应用中经常会被修改的文件类型制作而成的。文中将此类 iso、doc、html、pdf 等文件类型构成的数据集称为易变数据集。

4.2 BCCA 性能测试

实验主要比较以下三种不同的 Chunking 算法的性能：Rabin 算法、BCCA-P 和 BCCA-T。

BCCA 将 Chunking 划分为两个步骤：第一个步骤为预处理；第二个步骤为二进制串匹配，即 Chunking。

数据分块预处理就是在调用 Chunking 模块进行分块前,先处理一遍输入的正文数据,提取每个正文字节的某一个比特位数据,生成正文数据的一个位串特征数据,为后续的 Chunking 提供依据。

针对易变测试数据集,实验首先对 BCCA-T 和 BCCA-P 的预测处理时间与 Chunking 的时间比值进行了比较,其时间开销比值如图 9 和图 10 所示。

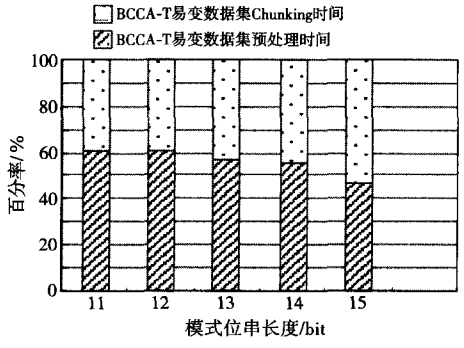

Figure 9　Time overhead ration of chunking to preprocessing using BCCA-T variable test data set

图 9　BCCA-T 易变测试数据集 Chunking 时间开销与预处理时间开销比例图

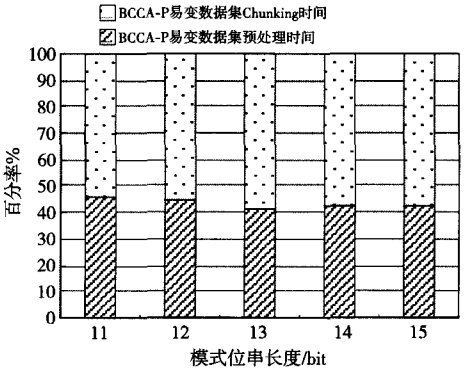

Figure 10　Time overhead ration of chunking to preprocessing using BCCA-P variable test data set

图 10　BCCA-P 易变测试数据集 Chunking 时间开销与预处理时间开销比例图

图中 X 轴和 Y 轴分别表示目标模式位串长度和不同处理时间所占总时间的百分率。实际上,对于同样的测试数据集,BCCA-T 与 BCCA-P 所用的预处理时间是相同的,但其所用的 Chunking 时间却不同。从图 9 和图 10 可以发现,根据设置的模式位串值的不同,BCCA-T 中预处理的时间占总时间的 47%～60%,而 BCCA-P 的预处理时间则大概占用了总时间的 42%～45%左右,这是一个相对来说比较大的时间开销。为此,BCCA 改进了这种实现,将预处理操作在一个预处理节点进行处理,Chunking 操作在前端服务器节点进行,通过并行操作实现了近似于流水线的工作模式。这样,进行大数据量的处理工作时,仅需一个启动的预处理时间,流水线工作正常后,数据实际处理时间就是 Chunking 的时间,从而可以大大减小实际时间开销。

接着,分别测试模式位串的不同长度时(其中,11 bit、12 bit、13 bit 和 14 bit 分别对应预期 Chunk 长度为 2 KB、4 KB、8 KB 和 16 KB),针对易变测试数据集获取的 Chunk 数量、冗余率和 Chunking 速度。实验分为五次处理,每次处理易变数据集中的一个子数据集,总共处理的数据为 50 GB 左右。

定义:利用某种 Chunking 算法,将一个数据对象划分为 n 个不重叠的数据块,称冗余 Chunk 的总数与总的 Chunk 块数的比值为 Chunk 冗余率 R,即:

R = 冗余 Chunk 总块数 /Chunk 总块数 × 100%

如图 11 所示,图中横轴表示 5 个子数据集,纵轴表示 Chunking 后产生的 Chunk 数量;图 12 中,图中横轴表示 5 个子数据集,纵轴表示 Chunking 后的 Chunk 冗余率。

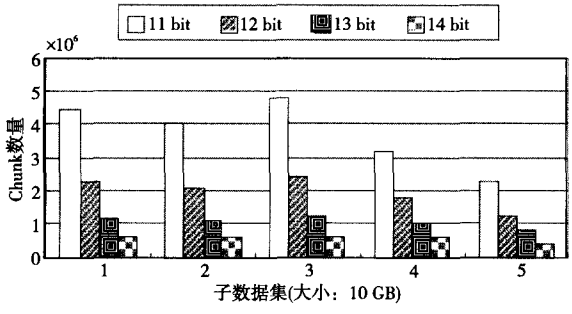

Figure 11　Comparison of the chunk numbers with different lengths of *Pattern*

图 11　采用不同长度的模式位串划分 Chunk 数量比较

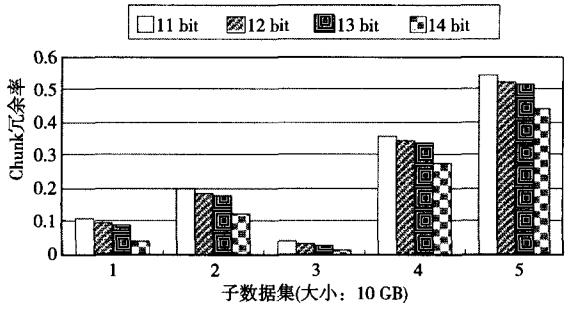

Figure 12　Comparison of chunk redundancy rate with different lengths of *Pattern*

图 12　采用不同长度的模式位串划分的 Chunk 冗余率比较

可以观察到,最终产生的 Chunk 数量与模式位串的长度紧密相关。随着模式长度的增大,所得

到的 Chunk 长度增加,每个子数据集 Chunking 后产生的 Chunk 数量按比例减少。当选用模式长度为 13 bit(对应 Chunk 预期长度为 8 KB)时,其产生的 Chunk 数量仅仅为模式长度为 11 bit(对应 Chunk 预期长度为 2 KB)时的 25%～35%左右,而 Chunk 的冗余率仅仅降低了 1.6%～2.5%左右。但是,如果继续增大模式长度到 14 bit(对应 Chunk 预期长度为 16 KB)时,Chunk 数量确实仍然按比例减少,但发现其 Chunk 冗余率却急剧减少,比模式长度为 11 bit 时几乎减少了 3%～10%左右。

图 13 显示了选择不同模式位串长度时,BCCA-T、BCCA-P 和 Rabin 三种不同的 Chunking 处理方式针对易变测试数据集的 Chunking 速度。图 13 中横轴表示选择的模式位串长度,纵轴表示 Chunking 速度。

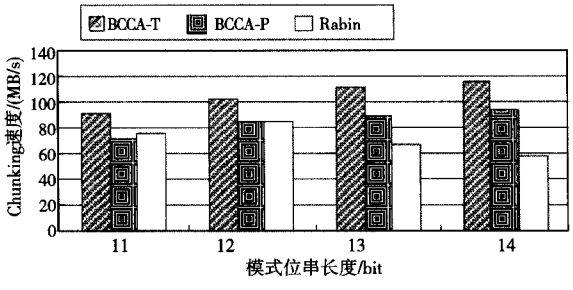

Figure 13 Comparison of Chunking speed of the variable test data set

图 13 易变测试数据集 Chunking 速度比较

当模式位串长度为 12 bit 时,Rabin 达到其最大的 Chunking 速度,为 84.9 MB/s 左右;而当模式位串长度为 14 bit 时,BCCA-T 与 BCCA-P 达到了其最大的 Chunking 速度,分别为 116 MB/s 和 93 MB/s 左右。这是由于 BCCS-P 采用了模式位串为匹配主体的思路,每次不匹配跳跃的距离有限,最大不会超过其位串的长度,获取的好处有限;而 BCCS-T 采用了正文位串为匹配主体的思路,利用每一次失败匹配尝试所带来的位特征信息,最大限度地排除不能匹配的位置,从而获得最大的跳跃长度,减少中间计算和比较的开销,从而减小了 Chunking 过程中数字签名计算的开销,达到提高系统性能的目的。因此,在选择相同模式位串长度的时候,BCCA-T 的 Chunking 速度几乎是 BCCA-P 的 1.2 倍左右,因此,相同情况下,BCCA-T 具有更高的效率。而相对于 Rabin 方式,当模式位串为 12 bit 时,BCCA-T 的 Chunking 速度较 Rabin 提高了 20%左右;当模式位串为 14 bit 时,BCCA-T 的 Chunking 速度较 Rabin 提高了 97%左右。由此可见,随着模式位串长度的增加,BCCA-T 能极大地提高 Chunking 的速度,但同时也会降低发现 Chunk 冗余的效率。因此,不能一味追求 Chunking 的速度,还必须同时考虑到 Chunk 冗余率。

5 结束语

在基于重复数据删除的容量缩减技术中,如何快速稳定地将待存储的数据对象划分为大小适合的 Chunk,是发现冗余数据和提高去重率的先决条件。已有的 Chunking 算法大多是通过 Rabin 指纹算法划分 Chunk,耗费大量的 CPU 计算资源,当需要处理的数据量变大时,数据指纹计算的 CPU 开销将大大增加,Chunking 算法将面临巨大的性能压力。

本文提出了一种新的基于位串内容感知的数据分块算法 BCCA。BCCA 从两个方面着手,首先,借鉴 Rabin 指纹算法中的增量指纹计算方法,根据二进制位串自身的特点,将原本需要大量 CPU 计算资源的指纹计算简化成了简单的移位运算;其次,将文件数据稳定分块的问题转换为两个二进制位串匹配的过程。根据匹配主体的不同,提出了两种不同的基于位串内容感知的分块策略 BCCA,即 BCCA-P 和 BCCA-T,而 BCCA-T 比 BCCA-P 能够更多地减少中间计算和比较的开销,从而更大地减小了 Chunking 过程中数字签名计算的开销,达到提高系统性能的目的。

最后,通过实验验证了基于位串内容感知的数据分块算法的优越性。

参考文献:

[1] You L, Pollack T, Long E. Deep store: An archival storage system architecture[C]// Proc of the 21stth International Conference on Data Engineering (ICDE'05), 2005: 804-815.

[2] Guo L, Efstathopoulos P. Building a high-performance deduplication system[C]// Proc of the 2011 USENIX Annual Technical Conference (USENIX'11), 2011: 25.

[3] Meyer D, Bolosky J. A study of practical deduplication[C]// Proc of the 9th USENIX Conference on File and Stroage Technologies (FAST'11), 2011: 1-14.

[4] Xie F, Condict M, Shete S. Estimating duplication by content-based sampling[C]// Proc of the 2013 USENIX Conference on Annual Technical Conference (ATC'13), 2013: 181-186.

[5] Ao Li, Shu Ji-wu, Li Ming-qiang. Data deduplication techniques[J]. Journal of Software, 2010, 21(5): 916-929. (in Chinese)

[6] Sun Ai-ling, Ran Lu-chun. Design and implementation of a data deduplication-based network file backup system[J]. Computer Applications and Software, 2014, 31(10): 86-90. (in Chinese)

[7] Zheng Ya-guang, Pan Jiu-hui. A duplicate data detection algorithm based on sliding blocking[J]. Computer Engineering, 2016, 42(2): 38-44. (in Chinese)

[8] Muthitacharoen A, Chen B, Mazières D. A low-bandwidth network file system[C]// Proc of Symposium on Operating Systems Principles (SOSP'01), 2001: 174-187.

[9] Min J, Yoon D, Won Y. Efficient deduplication techniques for modern backup operation[J]. IEEE Transactions on Computers, 2011, 6(6): 824-840.

[10] Jain N, Dahlin M, Tewari R. Taper: Tiered approach for eliminating redundancy in replica synchronization[C]// Proc of the 4th Conference on USENIX Conference on File and Storage Technologies (FAST'05), 2005: 21.

附中文参考文献:

[5] 敖莉,舒继武,李明强. 重复数据删除技术[J]. 软件学报, 2010, 21(5): 916-929.

[6] 孙爱玲,冉禄纯. 一种基于重复数据删除的网络文件备份系统设计与实现[J]. 计算机应用与软件,2014,31(10):86-90.

[7] 郑亚光,潘久辉. 一种基于滑动分块的重复数据检测算法[J]. 计算机工程,2016,42(2):38-44.

作者简介:

周斌(1971-),男,湖南长沙人,博士,副教授,CCF 会员(E22554M),研究方向为大数据存储与处理。**E-mail**: Binzhou@mail.scuec.edu.cn

ZHOU Bin, born in 1971, PhD, associate professor, CCF member (E22554M), his research interests include big data storage and processing.

朱容波(1978-),男,湖北潜江人,博士,教授,研究方向为云计算和物联网。**E-mail**: rbzhu@mail.scuec.edu.cn

ZHU Rong-bo, born in 1978, PhD, professor, his research interests include cloud computing, and Internet of Things.

张莹(1973-),女,湖北武汉人,硕士,讲师,研究方向为科技英语。

ZHANG Ying, born in 1973, MS, lecturer, her research interest includes English for science and technology.