小变化量数据的高效传输算法

——转置矩阵位压缩算法

庄 浩[1], 彭 颖[2]

(1. 中南大学 资源环境与建筑工程学院, 湖南 长沙 410083;
2. 广东证券有限责任公司 电脑中心, 广东 广州 510600)

摘要: 在许多网络数据传输中, 数据的变化量是很小的. 目前普遍使用的算法是传输变化了的数据项, 而忽略未变化的数据项. 当数据变化量不大时, 传输的冗余位很多, 这导致传输速度慢. 作者提出的转置矩阵位压缩算法, 可以将变化位和非变化位清晰地分开, 在以位为单位的水平上进行压缩, 这样即使使用极为简单的行程编码压缩算法也可以取得极高的压缩率. 试验结果表明: 该算法的对小变化量数据的压缩效果很好; 用于处理工业控制系统状态参数、证券交易中的行情数据等时, 可缓解系统的网络负荷压力, 提高网络的传输效率.

关键词: 网络; 转置矩阵; 压缩算法

中图分类号: TP301.6 **文献标识码**: A **文章编号**: 1005-9792(2001)01-0005-04

在通常的网络通讯中, 主从两端的数据同步传输十分普遍. 1 个信息点上的 1 张表要和另一个信息点上的同步(如工业控制中的系统状态数据、证券业务中的行情数据等均属于这一种), 这类数据有 1 个显著的特点, 即都是在 1 个基数上有小幅度的波动. 在普遍使用的传输算法中都是将数据划分为变化量和未变化量, 在传输时, 只将变化了的数据同步. 然而, 在实际系统中, 表中的所有数据几乎都有微小的变化. 在这种算法中, 不仅要传送变化的数据, 而且要传送变化量的位置. 在极端的情况下(即当每个数据都有变化时), 该种算法不但没有降低数据量, 反而加大了需要传送的数据即它需要传送数据的位置信息, 导致该算法效率很低.

这种算法是很难使用简单的行程编码压缩算法. 通用的压缩算法只能获得大概 60% ~ 80% 的压缩率. 目前, 微处理器的处理能力不断增强, 而通讯速度便成了制约信息传输的因素. 对于有规律的数据应使用有针对性的压缩算法. 为此, 作者针对小变化量的数据传输提出了 1 种新方法 ——转置矩阵位压缩算法. 即将变化了的数据位和未变化的数据位区分开来, 使得压缩在以位为单位的水平上进行. 位(bit)是数据可以分解到的最小单位[1~6].

1 算法描述

在计算机中, 所有的信息包括字符、整型数、浮点数、逻辑量等均是以二进制位的方式表达和存放的. 数据的变化即表现为某些数据位的变化, 在小变化量或慢变化系统的情况下, 在一定时间间隔内位的变化是很少的.

1.1 数据变化量的提取与数据恢复

用 1 个矩阵来表达要传输的系统状态.

考虑系统的初始状态为 $\boldsymbol{A}$:

$$\boldsymbol{A}=\begin{vmatrix} a_{1,n-1} & a_{1,n-2} & \Lambda & \Lambda & a_{1,0} \\ a_{2,n-1} & 0 & & & M \\ M & & 0 & & M \\ M & & & 0 & M \\ a_{m,n-1} & a_{m,n-2} & \Lambda & \Lambda & a_{m,0} \end{vmatrix} \quad (1)$$

矩阵的每个元素代表 1 个位(bit), 矩阵的每一行向量 $\boldsymbol{a}_i=\{a_{i,n-1}, a_{i,n-2}\Lambda a_{i,0}\}$ 代表数据中的 1 条纪录.

收稿日期: 2000-05-15

作者简介: 庄 浩(1969-), 男, 中南大学博士.

在时间间隔 Δt 后, 系统的状态为 $\boldsymbol{A}'$:

$$\boldsymbol{A}'=\begin{bmatrix} a'_{1,n-1} & a'_{1,n-2} & \cdots & \cdots & a'1,0 \\ a'_{2,n-1} & 0 & & & M \\ M & & 0 & & M \\ M & & & 0 & M \\ a'_{m,n-1} & a'_{m,n-2} & \Lambda & \Lambda & a'_{m,0} \end{bmatrix} \quad (2)$$

$$\Delta\boldsymbol{A}=\boldsymbol{A}\oplus\boldsymbol{A}'=\begin{bmatrix} a_{1,n-1}\oplus a'_{1,n-1} & a_{1,n-2}\oplus a'_{1,n-2} & \cdots & \cdots & a_{1,0}\oplus a'_{1,0} \\ a_{2,n-1}\oplus a'_{2,n-1} & 0 & & & M \\ M & & 0 & & M \\ M & & & 0 & M \\ a'_{m,n-1}\oplus a_{m,n-1} & a'_{m,n-2}\oplus a_{m,n-2} & \Lambda & \Lambda & a_{m,0}\oplus a'_{m,0} \end{bmatrix} \quad (3)$$

其中: $\oplus$是异或操作, 即将前后 2 个状态按位异或, 可以得到变化了的位的位置. 在 $\Delta\boldsymbol{A}$ 中, 没有变化的位为 0, 变化了的位为 1.

在慢变化系统中, 数据的变化量不大, $\Delta\boldsymbol{A}$ 是一个稀疏矩阵. 用简单的方法便可以获得很高的压缩率. 在传输中只需传送压缩后的 $\Delta\boldsymbol{A}$.

要恢复数据, 只需对(3)式进行简单的变换:

$$\boldsymbol{A}\oplus\Delta\boldsymbol{A}=\boldsymbol{A}\oplus(\boldsymbol{A}\oplus\boldsymbol{A}')\Rightarrow\boldsymbol{A}\oplus\Delta\boldsymbol{A}=\boldsymbol{A}' \quad (4)$$

即用 $\Delta\boldsymbol{A}$ 和原状态 $\boldsymbol{A}$ 按位异或就可以得到 Δt 时间后的系统状态 $\boldsymbol{A}'$.

1.2 实际系统数据的考虑

在实际的系统中, 系统状态一般以一组整型数、浮点数来描述的. 为了得到更高的压缩效率, 下面研究小变化量对存储在存储器中的数据位影响.

整型数在计算机中是直接以二进制数的方式存储的. 一个用 m 个整型数(2 bytes)表示的系统状态 $\boldsymbol{A}$ 可以写成:

$$\boldsymbol{A}=\begin{bmatrix} a_{1,15} & a_{1,14} & \Lambda & \Lambda & a_{1,0} \\ a_{2,15} & 0 & & & M \\ M & & O & & M \\ M & & & 0 & M \\ a_{m,15} & a_{m,14} & \Lambda & \Lambda & a_{m,0} \end{bmatrix} \quad (5)$$

假设系统的各个参数变化量均小于 $2^k(0\leqslant k<15, k\in\boldsymbol{Z})$, 那么, $\boldsymbol{A}$ 阵中的每个行向量的变化均集中于第 0 到第 $k-1$ 位. 显然,

$$\Delta\boldsymbol{A}=\begin{bmatrix} 0 & \Lambda & 0 & a_{1,k-1}\oplus a'_{1,k-1} & \Lambda & a_{1,1}\oplus a'_{1,1} & a_{1,0}\oplus a'_{1,0} \\ 0 & \Lambda & 0 & a_{2,k-1}\oplus a'_{2,k-1} & \Lambda & a_{2,1}\oplus a'_{2,1} & a_{2,0}\oplus a'_{2,0} \\ M & & M & M & & M & M \\ 0 & \Lambda & 0 & a_{m-1,k-1}\oplus a'_{m-1,k-1} & \Lambda & a_{m-1,1}\oplus a'_{m-1,1} & a_{m-1,0}\oplus a'_{m-1,0} \\ 0 & \Lambda & 0 & a_{m,k-1}\oplus a'_{m,k-1} & \Lambda & a_{m,1}\oplus a'_{m,1} & a_{m,0}\oplus a'_{m,0} \end{bmatrix} \quad (6)$$

浮点数在计算机中的表达是较复杂的. 根据 IEEE 的标准, 1 个浮点数分为 3 部分: 符号位(sign bit)、指数部分(exponent)及尾数(mantissa). 不同长度的浮点数的内存映像如表 1 所示.

表 1 IEEE 的浮点数格式

	real* 4	real* 8	real* 10
BYTE 1	*SXXX XXXX*	*SXXX XXXX*	*SXXX XXXX*
BYTE 2	*XMMM MMMM*	*XXXX MMMM*	*XXXX XXXX*
BYTE 3	*MMMM MMMM*	*MMMM MMMM*	1*MMM MMMM*
BYTE 4	*MMMM MMMM*	*MMMM MMMM*	*MMMM MMMM*
…		…	…
BYTE *n*		*MMMM MMMM*	*MMMM MMMM*

其中: S 是符号位; X 是指数位; M 是尾数位, 即二进制小数位, 每一位代表 1 个 2 的负指数幂, 如 1/2, 1/4, 1/8 等. Real* 4 的浮点数表示为

$$f=(-1)^S\times(1+M)\times 2^{X-127} \quad (7)$$

例如, 浮点数 1.5 在内存中表达为

$$1.5=(-1)^0\times\left(1+\frac{1}{2}\right)\times 2^{127-127} \quad (8)$$

其内存映像如表 2 所示.

表 2 IEEE Real* 4 浮点数 1.5 的内存映像

BYTE 1	BYTE 2	BYTE 3	BYTE 4
SXXX XXXX	*XMMM MMMM*	*MMMM MMMM*	*MMMM MMMM*
0011 1111	1100 0000	0000 0000	0000 0000

设浮点数 f 有 1 个小的变化 $\delta(-1<\delta<1)$, 那么,

$$f\times(1+\delta)=[(-1)^S\times(1+M)\times 2^{X-127}]\times(1+\delta)=$$
$$(-1)^S\times 2^{X-127}\times[(1+M)\times(1+\delta)]=$$
$$(-1)^S\times 2^{X-127}\times(1+M+\delta+M\times\delta) \quad (9)$$

根据(9)式, 并考虑 IEEE 的浮点数格式规范, 可以得到:

a. 当 $-1<(M+\delta+M\times\delta)<1$ 时, 变化量不会影响指数位. 否则, 浮点数得指数位亦会受到影响.

b. 当 δ 是 2 的负指数幂时, 尾数 M 的变化较小. 否则, 尾数 M 将有很大的变化. 然而 δ 是 2 的负指数幂的概率很小.

显然, 相对于整型数, 小变化量对浮点数的内存映像的影响很大. 虽然 1 个由浮点数表达的系统状态的 $\Delta\boldsymbol{A}$ 类似于(6)式的稀疏矩阵, 但远不如整数列的整齐. 这种区别源自于 2 种数的表达方式: 整型数使用 2 的正次幂表示有效位, 而浮点数使用 2 的负次幂表达有效位. 这样, 变化量对内存映像的影响在

整型数格式中被减小了, 而在浮点数格式中却被放大了.

因此, 应将浮点数放大以保留适当的有效位, 并转换为整型数, 再传送. 这样, 可以有效地增加矩阵 $\Delta \boldsymbol{A}$ 的行向量间的相关性, 以提高编码效率.

由(6)式可知, 对于类似于行程编码一类的压缩算法采用, $\Delta \boldsymbol{A}$ 的转置矩阵可以获得更高的压缩效率. 所以, 应当针对 $(\Delta \boldsymbol{A})^{\mathrm{T}}$ 压缩而不是 $\Delta \boldsymbol{A}$.

2　数值实例

为了验证前面提出的算法, 用 1 个变化的浮点数组 $\boldsymbol{A}$ 来验证. 设 $\boldsymbol{A}$ 由 128 个浮点数组成, 其内存映像如表 3 所示.

表 3　原始数 $\boldsymbol{A}$ 的内存映像

地址偏移量	数组 $\boldsymbol{A}$ 的内存映像			
0000	2F 16 00 00	63 20 00 00	EE 16 00 00	1C 00 00 00
0010	57 17 00 00	53 09 00 00	47 1B 00 00	61 0E 00 00
0020	CB 0B 00 00	FC 09 00 00	BA 0E 00 00	C2 12 00 00
0030	4A 04 00 00	30 15 00 00	81 16 00 00	6A 0C 00 00
0040	73 18 00 00	38 0B 00 00	DE 0D 00 00	42 22 00 00
0050	19 22 00 00	D1 05 00 00	2A 08 00 00	AA 08 00 00
0060	D2 1D 00 00	25 0C 00 00	BB 0B 00 00	EB 09 00 00
0070	0F 1E 00 00	BF 26 00 00	82 1F 00 00	3E 1F 00 00
0080	02 0A 00 00	F6 09 00 00	46 04 00 00	FE 03 00 00
0090	AC 11 00 00	E8 11 00 00	7A 01 00 00	67 0C 00 00
00A0	69 14 00 00	97 07 00 00	A9 22 00 00	1D 04 00 00
00B0	E1 1B 00 00	33 02 00 00	F4 10 00 00	92 23 00 00
00C0	97 01 00 00	29 03 00 00	D0 19 00 00	08 01 00 00
00D0	DB 12 00 00	1E 08 00 00	3A 10 00 00	5D 01 00 00
00E0	2D 06 00 00	69 17 00 00	17 1C 00 00	17 1C 00 00
00F0	6A 11 00 00	89 06 00 00	E3 24 00 00	86 01 00 00
0100	78 14 00 00	BF 05 00 00	9F 16 00 00	5C 0A 00 00
0110	6D 0B 00 00	86 02 00 00	4D 1F 00 00	CB 21 00 00
0120	D2 05 00 00	BA 24 00 00	6D 16 00 00	31 09 00 00
0130	F5 16 00 00	D1 02 00 00	1A 13 00 00	12 22 00 00
0140	74 03 00 00	85 09 00 00	B8 02 00 00	FD 02 00 00
0150	68 0A 00 00	48 1B 00 00	9F 1D 00 00	AF 1E 00 00
0160	4E 1D 00 00	64 1C 00 00	D4 0B 00 00	5E 18 00 00
0170	89 1D 00 00	10 08 00 00	BD 1D 00 00	20 1E 00 00
0180	67 20 00 00	DE 24 00 00	18 21 00 00	C5 1C 00 00
0190	2D 1C 00 00	15 06 00 00	59 20 00 00	CD 18 00 00
01A0	DA 21 00 00	B3 24 00 00	75 1B 00 00	59 02 00 00
01B0	0D 11 00 00	C7 02 00 00	FC 21 00 00	B7 05 00 00
01C0	D2 10 00 00	41 26 00 00	70 00 00 00	AB 23 00 00
01D0	7C 1E 00 00	2F 09 00 00	D8 0F 00 00	11 13 00 00
01E0	46 24 00 00	CF 0F 00 00	69 0F 00 00	AE 0B 00 00
01F0	8A 21 00 00	AD 17 00 00	62 08 00 00	77 20 00 00

将 $\boldsymbol{A}$ 的每一个数据加上 10% 的变化量, 求得 $\Delta \boldsymbol{A}$ 后对 $\Delta \boldsymbol{A}$ 乘上 100 并取整数部分, 即保留 2 位小数. 转置后得到的 $(\Delta \boldsymbol{A})^{\mathrm{T}}$ 的内存映像如表 4 所示.

表 4　原始数据 $(\Delta \boldsymbol{A})^{\mathrm{T}}$ 的内存映像

地址偏移量	数组 $\boldsymbol{A}$ 的内存映像			
0000	71 6B A4 B0	1A 84 C4 74	D0 10 A8 9F	56 F8 2D FB
0010	36 F0 F4 CA	F1 9D 8F 74	9A A8 A8 EF	E1 38 45 7A
0020	60 9E 1B B5	11 72 36 84	BA C3 AD 6D	86 FC 63 75
0030	94 9B 96 1C	9B 45 50 4E	E7 0C 83 D4	6D 2C DC 0A
0040	45 8B 9E 47	22 CD 94 41	AD 4E 74 E6	80 74 ED BD
0050	77 F0 EB 1B	AB 84 50 2F	E1 99 59 76	89 41 98 1D
0060	92 FA 56 54	AA C2 74 3A	AA 90 C2 D1	8D 81 71 C9
0070	83 EE 83 F0	18 D5 34 78	B5 52 90 0B	4B 56 EA 21
0080	97 6E 80 00	18 84 24 16	55 86 20 81	5B 03 6A B0
0090	91 02 00 80	18 04 24 46	81 04 20 00	11 41 42 20
00A0	10 00 00 00	08 00 20 06	01 00 20 00	11 00 40 20
00B0	10 00 00 00	00 00 20 02	00 00 00 00	01 00 40 20
00C0	00 00 00 00	00 00 00 00	00 00 00 00	01 00 40 00
00D0	00 00 00 00	00 00 00 00	00 00 00 00	01 00 00 00
00E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00F0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0100	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0110	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0120	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0130	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0140	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0150	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0160	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0170	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0180	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0190	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01A0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01B0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01C0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01D0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
01F0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

显然, 经过变换后, $(\Delta \boldsymbol{A})^{\mathrm{T}}$ 的前半部分和后半部分可以截然地区分开来. 前半部分集中反映的是数据变化的位, 而后半部分反映的是未变化的位. $\boldsymbol{A}$ 和 $(\Delta \boldsymbol{A})^{\mathrm{T}}$ 均经过 Huffman 编码压缩算法压缩后, 其大小分别是原来的 72.9% 和 47.7%. 可见, 该算法的压缩效果非常明显.

3　结　论

a. 该算法提取位的变化信息, 使压缩在数据位(Bit)的水平上进行. 通过数值实例证实, 使用该算法可以取得很高的压缩效率.

b. 相对于传统的算法, 该算法要存储原始的数

据 A 及变化后的数据 A' 和 $(\Delta A)^T$, 对计算机内存的需求虽大, 但这对目前的计算机硬件而言, 是很容易得到满足的.

参考文献:

[1] 王 嘉, 余松煜. 一种改进的基于零树集合的小波图像压缩算法[J]. 数据采集与处理, 2000, 15(1): 18-22.

[2] 白宜诚, 陆旭兵. 分布集中式微分测深数据采集系统[J]. 中南工业大学学报(自然科学版), 2000, 21(2): 184-187.

[3] 赵德平, 朱伟勇, 苏 畅, 等. 数据压缩字典与快速图像分形映射压缩算法[J]. 东北大学学报(自然科学版), 2000, 21(1): 147-152.

[4] 毛羽刚, 张拥军, 余士尧. 分布实时系统中的点到点通信[J]. 小型微型计算机系统, 2000, 21(2): 190-196.

[5] 傅祖芸. C 语言数值算法大全(第二版). 北京: 电子工业出版社, 1995.

[6] Steinmetz R, Nahrstedt K. Multimedia: computing, comunications and applications[M]. 北京: 清华大学出版社, 1996.

An efficient compress algorithm for slow varying system

ZHUANG Hao[1], PENG Ying[2]

(1. College of Resources Environment and Civil Engineering, Central South University, Changsha 410083, China;
2. Computer Center of Guangdong Securities Company, Guangzhou 510600, China)

Abstract: With the rapid development of computer hardware, network bandwidth becomes the bottleneck in network data transfer. A new kind of data compress algorithm was drawn in this paper. This algorithm is specially designed for slow varying systems, such as industry control system state parameters or stock prices in stock exchange. It separates data into varied and unvaried bits and forms a new variation matrix. The compress target is the transposed variation matrix. Both forward and reverse coding algorithms are discussed. The digital sample proves that this kind of compress algorithm is very efficient. The only disadvantage of this kind of algorithm is its higher memory requirement. But it is easy to meet in nowadays hardware environment.

Key words: network; variation matrix; compress algorithm