Compresión de Bases de Datos

M. C. Carlos Artemio Coello Coello

Escuela de Ingeniería Civil
Universidad Autónoma de Chiapas
Boulevard Belisario Domínguez km. 1081
Apartado Postal 61
C.P. 29000
Tuxtla Gutiérrez, Chiapas
México

Teléfono : **52+(961) 5-03-22**
Fax : **52+(961) 5-05-27**

E-mail : **coello@eecs.tulane.edu**

Ing. Héctor Ricardo Hernández de León

Instituto Tecnológico de Tuxtla Gutiérrez
Carretera Panamericana km. 1080
Tuxtla Gutiérrez, Chiapas
México

Teléfono : **(52)+(961) 6-26-55**

Compresión de Bases de Datos

Carlos Artemio Coello Coello
Universidad Autónoma de Chis.
Boulevard Belisario Dguez km. 1081
Tuxtla Gutiérrez, Chiapas (México)
coello@rex.cs.tulane.edu

Héctor Hernández de León
Instituto Tecnológico de Tuxtla Gutz.
Carretera Panamericana km. 1080
Tuxtla Gutiérrez, Chiapas (México)

RESUMEN

En este trabajo se aborda el problema del manejo económico de la información, haciendo énfasis en las bases de datos. Se dará una breve introducción a algunos de los conceptos fundamentales de la compresión sin pérdida de datos, acompañados de un análisis igualmente breve de las técnicas más populares en la actualidad y de una comparación de su eficiencia al aplicarse a tipos diferentes de archivos. Finalizaremos describiendo algunas de las áreas abiertas de investigación que esperan a ser abordadas por los interesados en el tema.

Introducción

Indudablemente el exceso de información ha sido y seguirá siendo un problema al que deberemos enfrentarnos en una sociedad que es cada vez dependiente de la tecnología. Pese a que en la actualidad los costos de almacenamiento se han visto reducidos de una forma dramática, éste sigue siendo un problema a tomar en cuenta, especialmente para aplicaciones de gran tamaño. Dos son las razones principales por las que es importante compactar la información de un sistema de base de datos [1]: el ahorro de espacio y las mejoras a la eficiencia del sistema. La primera razón es obvia, y la segunda se deriva del hecho de que como es bien sabido, resulta más eficiente manipular cantidades pequeñas de información. Sin embargo, hay también algunas desventajas implícitas [2]: se requiere tiempo adicional de procesamiento para realizar la compresión y descompresión de la información; algunos métodos de compresión producen registros de longitud variable que son más difíciles de almacenar y recuperar; algunas técnicas requieren efectuar una manipulación de bits, lo cual suele ser difícil si se usa un lenguaje de alto nivel; se requiere también tiempo adicional para análisis, diseño, programación y verificación durante el desarrollo del nuevo sistema y, finalmente, se requiere algunas veces efectuar el mantenimiento de las tablas de codificación y decodificación conforme se almacenan nuevos valores en la base de datos. Sin embargo, en ciertas condiciones parece ser altamente rentable la utilización de alguna técnica de compresión de datos, y en otros, resulta incluso la única solución. La organización de este trabajo es la siguiente: comenzaremos con algunas

generalidades sobre compresión de datos, después describiremos brevemente las técnicas más conocidas para compresión de texto mencionando sus ventajas y desventajas. Posteriormente mostraremos un estudio comparativo de dichas técnicas al aplicarse a diferentes tipos de archivos. Finalizaremos describiendo nuestro trabajo futuro y algunas de las áreas abiertas de investigación.

Nociones de Compresión de Datos

Desde que Claude Shannon estableció los principios de la Teoría de la Información a fines de los 40s [3], los investigadores empezaron a interesarse en la forma de comprimir datos. Presumiblemente todos los mensajes poseen información redundante de la cual podemos deshacernos, reduciendo así su tamaño. La Teoría de la Información usa el término entropía para medir la cantidad de información codificada en un mensaje. Esta palabra fue tomada de la termodinámica, y tiene un significado similar. A mayor entropía de un mensaje, mayor información contiene. La entropía de un símbolo se define como el logaritmo negativo de su probabilidad. Para determinar el contenido de información de un mensaje en bits, expresamos la entropía usando la expresión:

$$\text{Número de bits} = -\text{Log}_2\,(\text{probabilidad})$$

La entropía de un mensaje es simplemente la suma de las entropías de cada uno de sus símbolos. La entropía encaja en la compresión de datos porque nos permite determinar los bits de información presentes en un mensaje, permitiéndonos saber entonces cuánto puede comprimirse éste. Por ejemplo, si la probabilidad de la letra 'a' en este documento es 1/32, entonces el contenido de información de esta letra es 5 bits. De tal forma, la cadena "aaaaaaaaaa" tiene un contenido de 50 bits. Si embargo, si usamos caracteres ASCII de 8 bits para codificar esta cadena, se requerirán 80 bits para almacenarla. Esto nos da una idea del ahorro de espacio que podemos lograr si elegimos el esquema apropiado de almacenamiento. Si la redudancia es la clave en la compresión de datos, lo lógico es tratar de explotar las diferentes formas en que ésta se produce en las bases de datos. Actualmente, pueden distinguirse 4 tipos básicos de redundancia en las bases de datos [4]:

- Distribución de Caracteres: Se sabe que algunos caracteres tienden a aparecer con más frecuencia que otros en los textos de un cierto idioma. Por ejemplo, la letra 'e' es la más usada en inglés, y la 'a' es la más usada en español. Si usamos el formato ASCII de 8 bits para almacenar nuestra base de datos, todos los caracteres consumirán 8 bits. Sin embargo, si usamos un esquema distinto, podemos valernos de una representación que consuma sólo 1 ó 2 bits para representar la letra 'a', produciéndose así un ahorro considerable de espacio. Este es el enfoque usado en técnicas como la **codificación de Huffman** [5]. Este método es uno de los más populares en la práctica, por su facilidad de implemen-

tación, ya que cada caracter se codifica de acuerdo a su probabilidad de aparición en la base de datos. La tabla de búsqueda que se requiere es muy económica, pero el grado de compresión puede verse limitado por el número máximo de entradas de la misma. Uno de los principales problemas de esta técnica es la complejidad del proceso de descompresión debido a que no se sabe de antemano la longitud de cada código sino hasta que se descompactan los primeros bits. En general, el método de Huffman se encuentra en seria desventaja cuando el volumen de datos es alto. Adicionalmente, debe conocerse la distribución de frecuencias de los símbolos de entrada de antemano, lo que puede implicar problemas en algunos esquemas de almacenamiento.

- Repetición de Caracteres: En bases de datos científicas y numéricas es frecuente que existan repeticiones consecutivas de un mismo caracter dentro de una cadena. Cuando se almacenan imágenes en formato binario, estos patrones suelen ser todavía más frecuentes. Este esquema amerita el uso de un método que permita codificar una cadena a través de un simple par de datos: el caracter y el número de veces que éste se repite. Este es el principio usado por una técnica que se conoce como **codificación de longitud de ejecución** (run-length encoding). Esta técnica ha sido usada con relativo éxito en gráficos (por ejemplo, en el formato .PCX), aunque no suele resultar muy efectiva con archivos de texto. Sin embargo, en [2] se presenta evidencia de que la supresión de los espacios en blanco, los ceros y otros caracteres de relleno usados para rellenar la longitud de los campos de una variable en algunos sistemas de bases de datos comerciales permite compresiones del orden del 75%.
- Patrones de uso frecuente: En este caso la idea es similar al caso anterior, pero ahora se buscan patrones de 2, 3 ó más caracteres. De forma análoga, se sustituirán estos patrones por representaciones que requieran menos bits, produciéndose así excelentes resultados. Puede usarse una lista previa de patrones comunes en el lenguaje determinado -a estos métodos se les llama de **diccionario**- o bien puede generarse dicha lista en tiempo de ejecución y modificarse conforme se recorre el archivo -esta es la base del método **LZW** [6](Lempel, Ziv y Welch, los apellidos de sus creadores)-. Ambos enfoques pueden resultar útiles dependiendo de las circunstancias y la disponibilidad de recursos de hardware. Uno de los problemas del método LZW es su pobre desempeño durante la porción inicial de un mensaje; como consecuencia, el mensaje debe ser lo suficientemente largo para que la compresión resulte efectiva.
- Redundancia Posicional: Ciertos caracteres o patrones pueden aparecer consistentemente en un lugar predecible en cada bloque de datos. Esto ocurre, por ejemplo, en datos de rastreo o cualquier tipo de estructura de datos preformada donde ocurra una cierta representación en la misma posición a cada cierto intervalo de tiempo. La **Compresión Diferencial** cae dentro de esta categoría, y suele utilizarse en aplicaciones donde los datos son de tamaño uniforme y tienden a variar relativamente despacio, tal y como en el caso de los datos de telemetría. Esta técnica, sin embargo, suele resultar inoperante en bases de datos comerciales, debido a que los patrones de caracteres en los que se basa ocurren muy raramente en ellas.

De los métodos mencionados anteriormente, el más utilizado en la práctica es el de Huffman, seguido por el LZW. Durante los últimos 10 años sólo ha surgido una contrincante

serio a ellos: la **Codificación Aritmética** [7]. Este método representa un mensaje mediante un intervalo de números reales entre 0.0 y 1.0. Conforme el mensaje se vuelve más largo, el intervalo requerido se vuelve más pequeño y, en consecuencia, el número de bits necesarios para especificarlo crece. El mayor inconveniente de este método es precisamente sus tremendas demandas de memoria, pero su eficacia está fuera de toda discusión como se verá más adelante.

Detalles de Implementación y Medición

Para nuestras pruebas usamos código de dominio público contenido en [8] y encontrado en el Internet, el cual hubo de ser adecuado a nuestras necesidades específicas. Además de los métodos antes mencionados se experimentó con técnicas híbridas (i.e., que combinan más de un método). En vez de limitarnos a usar la razón de compresión[1] para medir la eficiencia de cada técnica, usamos varios factores de medición, incluyendo la denominada 'ganancia de compresión' (compression gain), propuesta por Paul Howard [9]:

$$\text{Ganancia de Compresión} = 100\log_e \frac{t}{l}$$

donde: t=tamaño del archivo original en bytes.
l=tamaño del archivo compactado en bytes.

Asimismo analizamos diferentes tipos de archivos: listados de programas en C y Pascal, números de coma flotante, documentos en español almacenados en ASCII y otros formatos comunes de procesamiento de texto, bases de datos creadas con sistemas comerciales (e.g. DBase y Paradox), datos científicos formateados y archivos ejecutables y de sistema.

Comparación de Resultados

Debido a las limitantes de espacio sólo podremos incluir algunas de las tablas de comparación de los datos obtenidos con nuestras pruebas (ver Tablas 1 y 2). Estos valores son, sin embargo, bastante representativos del comportamiento de los métodos en estudio. En general, la técnica híbrida usada por el **Sixpack** parece ser la que muestra un mejor desempeño, excepto en el caso de archivos de gran tamaño. El método más lento fue el **Mix-1**, aunque el más ineficiente en términos de compresión efectiva fue el método de **compresión diferencial**. Uno de los puntos interesantes que debe observarse en estas tablas es el hecho de que en algunos casos se obtienen archivos más grandes del tamaño original debido a la ineficiencia del método. Contrario a lo que pudiera pensarse, los métodos híbridos no son siempre los mejores, aunque en algunos casos parece haber cierta rela-

[1] $\text{RC} = \frac{\text{Tamaño del archivo sin comprimir}}{\text{Tamaño del archivo comprimido}}$

ción entre tiempo de compresión y eficiencia del método. No obstante, existen otros factores involucrados como son la cantidad de memoria disponible (un punto clave en los métodos de codificación aritmética) y el tamaño del archivo fuente. Brevemente podemos decir que la relación **T/L** nos permite determinar en cuántas veces se redujo el tamaño del archivo, la **razón de compresión** nos da el mismo valor en forma de porcentaje, y la **ganancia de compresión** lleva los resultados a una escala logarítmica a fin de hacerlos más fáciles de comparar.

Método	Tam. Original[2]	Tam. Comp.	T/L	Razón de Comp.	100 Log_e(T/L)	Tiempo (seg)
Huffman	13224063	4725562	2.80	64.27%	102.91	118.96
Huffman-A[3]	13224063	2691246	4.91	79.65%	159.20	174.48
Cod. Aritm 0[4]	13224063	4651746	2.84	64.82%	104.48	552.99
LZSS	13224063	2364813	5.59	82.12%	172.13	460.05
LZW 12 bits	13224063	18141837	0.73	-37.19%	-31.62	314.45
Diferencial	13224063	4595543	2.88	65.25%	105.70	816.24
Sixpack	13224063	N.D.	N.D.	N.D.	N.D.	N.D.
Cod. Aritm 1[5]	13224063	876789	15.08	93.37%	271.35	4781.54
Mix-1 0	13224063	2118179	6.24	83.98%	183.15	25212.23

Tabla 1: *Resultados producidos al aplicar los métodos en estudio a un archivo en formato Microsoft Word para Windows 2.0. Las pruebas se corrieron en una computadora 486 DX/2 de 66 MHz con coprocesador matemático compatible con IBM. El método* ***Sixpack****, que consiste en una mezcla de 6 técnicas diferentes produjo un error de desbordamiento. El método* ***Mix-1*** *también es un híbrido que se probó únicamente en el nivel cero por su tremenda ineficiencia.*

Método	Tam. Original[6]	Tam. Comp.	T/L	Razón de Comp.	100 Log_e(T/L)	Tiempo (seg)
Huffman	504802	180317	2.80	64.28%	102.95	4.01
Huffman-A[7]	504802	169595	2.98	66.40%	109.08	7.69
Cod. Aritm 0[8]	504802	176729	2.86	64.99%	104.95	20.76
LZSS	504802	141623	3.56	71.94%	127.10	26.80
LZW 12 bits	504802	133938	3.77	73.47%	132.68	5.16
Diferencial	504802	312909	1.61	38.01%	47.83	53.28
Sixpack	504802	85210	5.92	83.12%	177.90	16.92
Cod. Aritm 1[9]	504802	122461	4.12	75.74%	141.64	107.71
Mix-1 0	504802	164523	3.07	67.41%	112.11	12950.99

Tabla 2: *Resultados producidos al aplicar los métodos en estudio a un archivo en formato .DBF. Las pruebas se corrieron en una computadora 486 DX/2 de 66 MHz con coprocesador matemático compatible con IBM.*

---

[2] En bytes.

[3] Huffman Adaptativo.

[4] Modelo de orden cero fijo usando codificación aritmética.

[5] Model adaptativo de orden n con codificación aritmética de orden 1.

[6] En bytes.

[7] Huffman Adaptativo.

[8] Modelo de orden cero fijo usando codificación aritmética.

[9] Model adaptativo de orden n con codificación aritmética de orden 1.

Areas Abiertas de Investigación y Conclusiones

Recientemente se ha realizado mucho trabajo en cuanto a desarrollo de hardware que soporte esquemas de compresión de datos. Por ejemplo, hay dispositivos que hacen la función de coprocesadores de compresión, aumentando de forma drástica la eficiencia de las técnicas tradicionales al efectuar vía hardware la creación de los árboles o las tablas de búsqueda y al realizar la búsqueda en ellas. De tal forma se ha podido lograr la compresión en tiempo real. Asimismo, se ha logrado una eficiencia en tiempo real muy razonable aún en la ausencia de hardware especializado, sacrificando un poco de memoria en sistemas comerciales como el altamente celebrado Stacker™. No obstante, se requiere todavía más trabajo en esta área. También se necesita investigar más en torno a la compresión de los metadatos (i.e., datos dentro de los datos), en torno a la explotación de relaciones semánticas en los datos. Más algoritmos debieran desarrollarse en vez de mantenerse reformando constantemente los ya escritos, y se requieren más estudios de redundancia y desempeño de los métodos existentes para el caso particular de nuestro idioma. Nosotros nos encontramos trabajando en esta última parte, con la esperanza de mejorar de forma significativa los métodos existentes cuando se apliquen a documentos en español, y con vistas a proponer al menos un método nuevo.

Referencias

[1] Roth, Mark A. and Van Horn, Scott J. Database Compression. *SIGMOD Record*, Vol. 22, No. 3, September 1993, pp. 31-39.

[2] Severance, Dennis G. A Practitioner's Guide to Data Base Compression. *Information Systems*, Vol. 8, No. 1, pp. 51-62, 1983.

[3] Shannon, Claude.A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948.

[4] Welch, Terry A. A Technique for High Performance Data Compression. *Computer*, pp. 8-19, June 1984.

[5] Huffman, David A. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE, 40:1098-1101*, September 1952.

[6] Ziv, J. & Lempel A. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Information Theory*, Vol. IT-23, nO. 3, mAY 1977, PP. 337-343.

[7] Witten, Ian H., et al. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6), pp. 520-540, June 1987.

[8] Nelson, Mark. **The Data Compression Book**. M&T Books. 1992. 527 p.

[9] Howard, Paul G. **The Design and Analysis of Efficient Lossless Data Compression Techniques**. Department of Computer Science. Brown University. Technical Report No. CS-93-28. 97 p. June 1993.