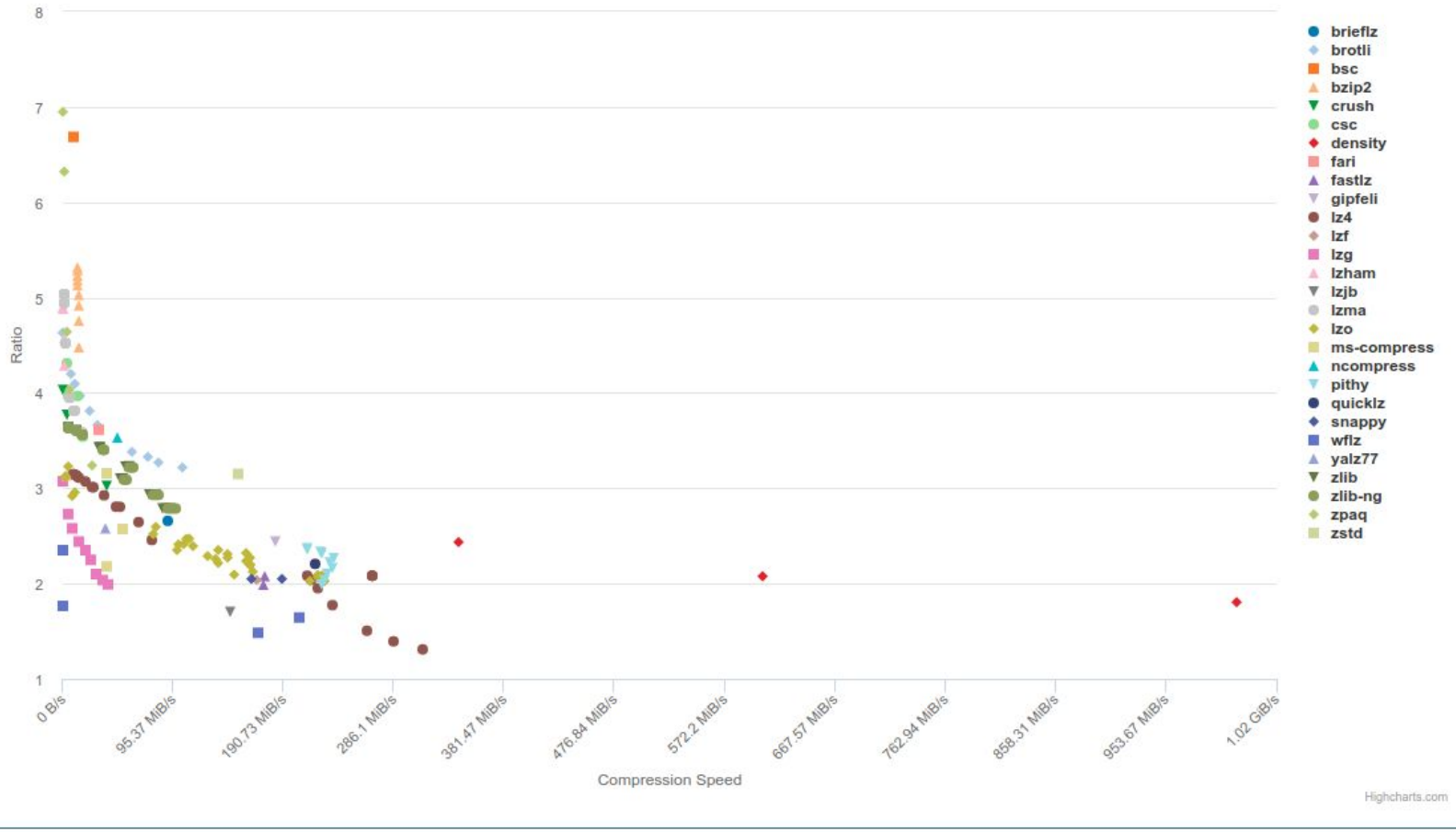
Compresión de Datos en la Era de Big Data

75.06 Organización de Datos

Temas

- La Frontera de Pareto
- Algunos compresores clásicos: DEFLATE
- Comprimiendo muy rápido: La familia LZ de Compresores
  - SNAPPY
  - LZ4
  - LZSE
- LZMA
- La Transformación de Burrows y Wheeler: Block Sorting
- El estado del arte en Compresión de Datos: PAQ

COMPRESSION RATIO VS. COMPRESSION SPEED



Algunas Familias Importantes

- Nivel de compresión
  - DEFLATE, BZIP, LZMA, PAQ
- Velocidad de compresión / descompresión
  - LZ4, LZSE, SNAPPY

La Familia LZ de Compresores

- Idea: Reemplazar secuencias repetidas de caracteres por un puntero a donde las vimos primero y una longitud.
- Los primeros algoritmos en trabajar este modelo fueron LZ77 y LZ78 creados por Lempel y Ziv en 1977 y 1978 respectivamente.

LZ77

LZ77

BIT FLAG:
    0 = CARACTER LITERAL (seguido del caracter)
    1 = REPETICION (seguido de un par OFFSET, LONGITUD)

Ejemplo:
    ABCDABCEABCD

    0A,0B,0C,0D,1,3,3,0E,3,3,0D   (una posible opción)

LZ78

LZ78

Usamos un diccionario (tabla) para almacenar repeticiones.
Originalmente las posiciones 0 a 255 contienen los 256 caracteres individuales (repeticiones de longitud 1)

Ejemplo:

ABCDABCEABCD

(A)(B)(C)(D)(256)(C)(E)(260)

256 = AB, 257 = BC, 258=CD, 259=DA, 260=ABC, 261=CE
262 = EA, 263 = ABCD

Buscando Repeticiones

- Buscar las repeticiones en el buffer es el paso más costoso de cualquier compresor tipo LZ. (La descompresión es en cambio instantánea)
- Pueden usarse diferentes búsquedas:
  - Fuerza bruta
  - Árboles (por ejemplo B-Trees o B+)
  - Hashing
- La búsqueda puede ser greedy o exhaustiva
- Más allá de que la búsqueda sea exhaustiva no siempre emitir una repetición es la estrategia óptima.

DEFLATE

- Es uno de los algoritmos de compresión más populares.
- Está basado en una simple combinación de LZ77 y Huffman.
- Es el algoritmo default de los compresores tipo Zip (Pkzip, Gzip, etc)

DEFLATE

DEFLATE: es un LZ en donde se usa un árbol de Huffman para codificar las longitudes y los caracteres literales.

Y otro árbol para codificar los offsets.

(obviamente con algunos detalles)

Ejemplo:
ABCDABCEABCD

[A],[B],[C],[D],[L3],[D3],[E],[L4],[D7]

Un árbol codifica A,B,C,D,E…,L2,L3,L4…
Otro árbol codifica D0,D1,D2…

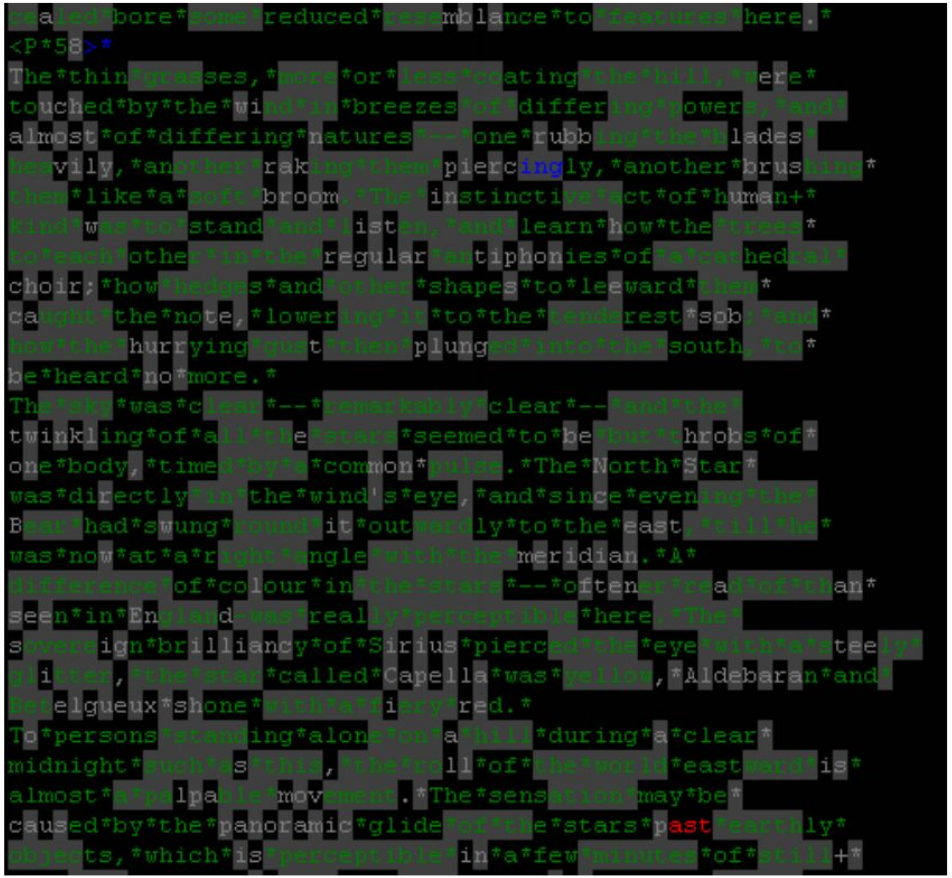
LZMA

- Es la versión más avanzada de la familia LZ.
- Usado por el compresor 7zip.
- Basado en la combinación de LZ y compresión aritmética.

LZMA

```
0 + char => LITERAL
1 + 0 + len + dist => MATCH
1 + 1 + 0 + 0 => SHORTREP (L=1, misma distancia)
1 + 1 + 0 + 1 + len => LONGREP[0] (dist = prev dist)
1 + 1 + 1 + 0 + len => LONGREP[1] (dist = prev prev dist)
1 + 1 + 1 + 1 + 0 + len => LONGREP[2] (etc..)
1 + 1 + 1 + 1 + 1 + len => LONGREP[3]
```

LZMA



Gris = LITERAL
Verde = MATCH
Azul = SHORTREP
Blanco = LONGREP0
Rojo = LONGREP1..3

Compresión Ultra Rápida con la Familia LZ

- Los algoritmos de la familia LZ son extremadamente rápidos para descomprimir. ¿Por qué?
- Con un poco de cuidado también se pueden programar para que sean eficientes al comprimir.
- No es necesario codificar cada símbolo usando Huffman o Aritmético, simplemente codificamos las repeticiones.
- ¡Conocer estos algoritmos es muy importante! Son fáciles de implementar y muy rápidos. Esto es ideal cuando no tenemos un compresor disponible en una biblioteca.

Snappy

- Es una biblioteca de compresión basada en la familia LZ desarrollada por Google [2011]
- Se usa en proyectos como BigTable, MapReduce, Cassandra, MongoDB, LevelDB, RocksDB y Lucene.
- El autor original:.....Jeff Dean

SNAPPY

LFILE_LEN (Admite números de longitud variable, lower 7 bits used for data)
TAG BYTE
First two BITS:
00 = LITERAL (upper 6 bits for length) (Admite l variable)
01 = COPY (length 3 bits, offset 11 bits)
10 = COPY (length 6 bits, offset 2 bytes)
11 = COPY (length 6 bits, offset 4 bytes)

LZ4

- LZ4 es otra variante de LZ desarrollado en 2011 por Yann Collet [Facebook]
- Se usa en varias partes de Linux, Hadoop, MySQL, Spark, etc.

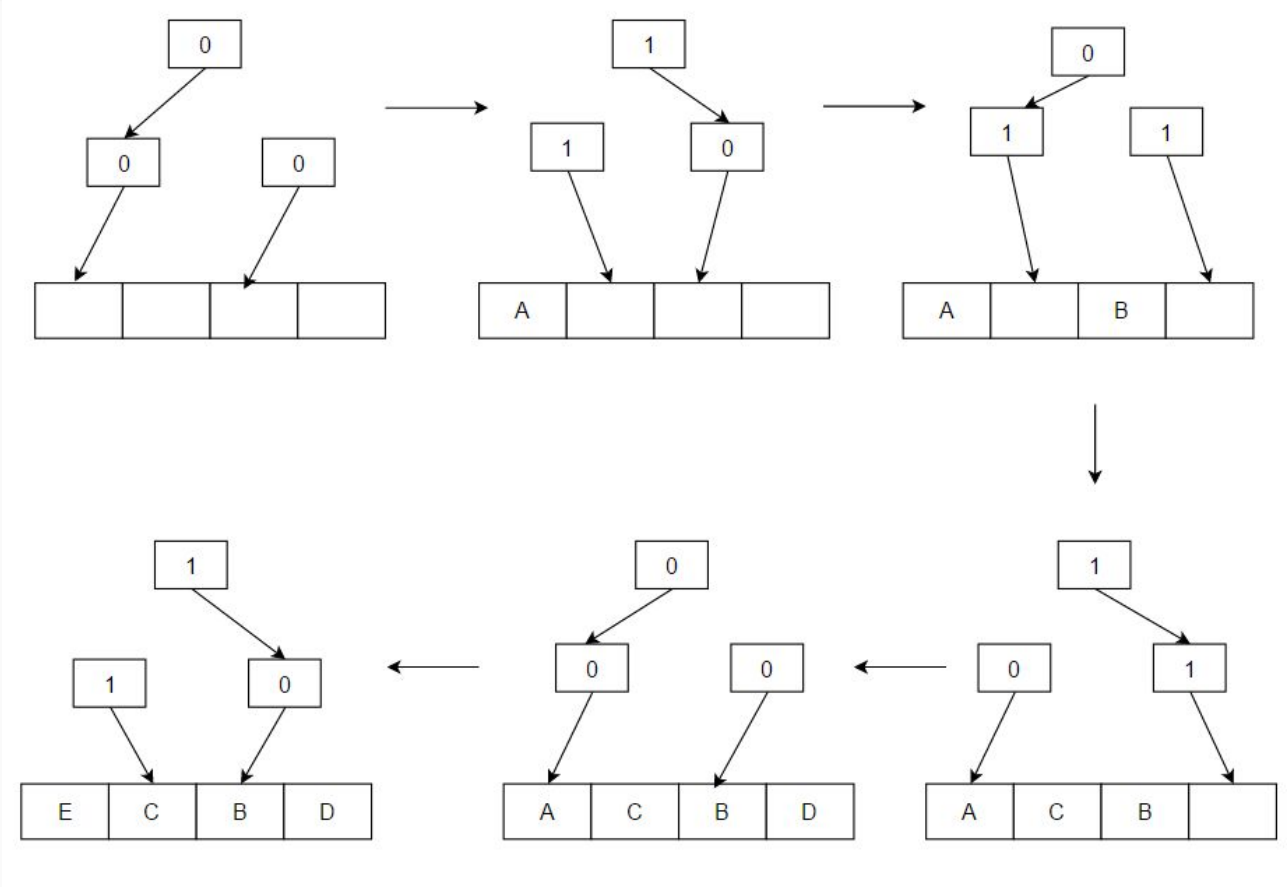
LZ4

```
FIRST BYTE: TOKEN
    LOWER 4 = T1
    HIGH 4 = T2
Next: e1 (Coded in LSIC if T1==15) (LSIC = 255+ encoding)
Next: t1 + e1 LITERALS
Next: OFFSET (2 bytes)
Next: e2 (Coded in LSIC if T2==15)
Copiar t2+e2+4 bytes from position -OFFSET
```

LZ4

- Para encontrar repeticiones usa un B-Tree o un Hash (4 caracteres)
- Para evitar que se llene la memoria usa una política de reemplazo de cache, concretamente PRU.

PRU



LZSE

- Idea: Usar Huffman (o similar) para codificar el resultado de un algoritmo tipo LZ.
- En general esto mejora el nivel de compresión pero empeora la velocidad de compresión y descompresión.
- Códigos Tunstall

Tunstall Codes

Ejemplo: AACABAABAADE (A=7/12, B=2/12, C=1/12,D=1/12,E=1/12)
Inicialmente 5 símbolos = 3 bits por símbolo
(Ej A=000,B=001,C=010,D=011,E=100)
Podemos expandir A qué es el símbolo más probable

```
A A   0000   (prob es 7/12 * 3/7) =0.25
  B   0001   (prob es 7/12 * 2/7) =0.16
  C   0010   (prob es 7/12 * 1/7) =0.0833
  D   0011   (prob es 7/12 * 1/7) =0.0833
B     0100   (prob es 2/12) = 0.16
C     0101   (prob es 1/12) = 0.0833
D     0110   (prob es 1/12) = 0.0833
E     0111   (prob es 1/12) = 0.0833
```

Tunstall Codes

Ejemplo: AACABAABAADE
Podemos expandir AA qué es el símbolo más probable

A	A	C	0000 (7/12*3/12*⅓)
	A	B	0001
	A	D	0010
	B		0011
	C		0100
	D		0101
B			0110
C			0111
D			1000
E			1001

etc….

Block Sorting

Block Sorting

- El algoritmo está basado en una combinación de tres pasos:
  - La transformación de Burrows y Wheeler
  - Move to Front
  - Un compresor Estadístico
- Además de servir como algoritmo de compresión en BZIP estos algoritmos tienen propiedades muy interesantes (Compression Indexing)

La Transformación de Burrows y Wheeler

Para cada bloque del archivo:

ABCDABCEABCD
BCDABCEABCDA
CDABCEABCDAB
DABCEABCDABC
ABCEABCDABCD
BCEABCDABCDA
CEABCDABCDAB
EABCDABCDABC
ABCDABCDABCE
BCDABCDABCEA
CDABCDABCEAB
DABCDABCEABC

La Transformación de Burrows y Wheeler

Para cada bloque del archivo:

ABCDABCDABC**E**
ABCDABCEABC**D** **<= STRING ORIGINAL (L=1)**
ABCEABCDABC**D**
BCDABCDABCE**A**
BCDABCEABCD**A**
BCEABCDABCD**A**
CDABCDABCEA**B**
CDABCEABCDA**B**
CEABCDABCDA**B**
DABCDABCEAB**C**
DABCEABCDAB**C**
EABCDABCDAB**C**

MOVE TO FRONT [MTF]

```
EDDAAABBBCCC
MOVE TO FRONT [ABCDE]
E = 4 [EABCD]
D = 4 [DEABC]
D = 0 [DEABC]
A = 2 [ADEBC]
A = 0 [ADEBC]
A = 0 [ADEBC]
B = 3 [BADEC]
B = 0 [BADEC]
B = 0 [BADEC]
C = 4 [CBADE]
C = 0 [CBADE]
C = 0 [CBADE]
```

ALGUN COMPRESOR ESTADISTICO

```
4,4,0,2,0,0,3,0,0,4,0,0

0 = 7/12 => 0.77 bits c/u
2 = 1/12 => 3.58 bits c/u
3 = 1/12 => 3.58 bits c/u
4 = 3/12 => 2 bits c/u

Total = 18.55 bits
```

MOVE TO FRONT [MTF] (Reversa)

```
4,4,0,2,0,0,3,0,0,4,0,0 [ABCDE]
4 = E [EABCD]
4 = D [DEABC]
0 = D [DEABC]
2 = A [ADEBC]
0 = A [ADEBC]
0 = A [ADEBC]
3 = B [BADEC]
0 = B [BADEC]
0 = B [BADEC]
4 = C [CBADE]
0 = C [CBADE]
0 = C [CBADE]
=> EDDAAABBBCCC
```

La Transformación de Burrows y Wheeler (Reversa)

EDDAAABBBCCC ordenando AAABBBCCCDDE
En el string de arriba
(A=3,A=4,A=5,B=6,B=7,B=8,C=9,C=10,C=11,D=1,D=2,E=0) **L=1**
A**A**AABBBCCCDDE (A=4)
AAAB**B**BCCCDDE (B=7)
AAABBBC**C**CDDE (C=10)
AAABBBCCCD**D**E (D=2)
AA**A**BBBCCCDDE (A=5)
AAABB**B**CCCDDE (B=8)
AAABBBCC**C**DDE (C=11)
AAABBBCCCDD**E** (E=0)
AAABBBCCCDDE (A=3)
AAA**B**BBCCCDDE (B=6)
AAABBB**C**CCDDE (C=9)
AAABBBCCC**D**DE (D=1) FIN => ABCDABCEABCD

El Estado del Arte en Compresión de Datos: PAQ

PAQ

- PAQ combina varios modelos de compresión diferentes (Context Mixing)
- Procesa el Archivo BIT a BIT
- Es extremadamente lento y consume muchísimos recursos
- Pero es el estado del arte en cuanto a nivel de compresión para casi todo tipo de archivo.

PAQ: Modelos

- N-Grams: Probabilidad de 1/0 según los N-bytes y N-bits anteriores.
- N-Grams x palabra: Probabilidad de 1/0 según las N palabras anteriores.
- Modelo especializado en tablas
- Modelo especializado en los bits superiores en los N bytes anteriores.
- (Según la versión puede haber otros modelos)

PAQ: Modelos

Cada modelo emite una probabilidad.

Los modelos se mezclan mediante una suma ponderada para determinar la probabilidad final.

La ponderación se hace mediante una red neuronal que va aprendiendo a medida que comprime (!)

COMPRESION ARITMETICA

Idea: Convertir la probabilidad en un cierto número de bits.

Ej: **1**0110 (supongamos 1=⅗, 0=⅖)

Intervalo inicial: [0...0.4)=0 [0.4..1)=1

=> Resultado = [0.4..1)

COMPRESION ARITMETICA

Idea: Convertir la probabilidad en un cierto número de bits.

Ej: **10**110 (supongamos 1=⅗, 0=⅖)

[0.4..0.64) = 0 [0.64..1) = 1

=> Resultado = [0.4..0.64)

COMPRESION ARITMETICA

Idea: Convertir la probabilidad en un cierto número de bits.

Ej: **101**10 (supongamos 1=⅗, 0=⅖)

[0.4..0.544) = 0 [0.544..0.64) = 1

=> Resultado = [0.544..0.64)

COMPRESION ARITMETICA

Idea: Convertir la probabilidad en un cierto número de bits.

Ej: **1011**0 (supongamos 1=⅗, 0=⅖)

[0.544..0.5825) = 0 [0.5824..0.64) = 1

=> Resultado = [0.5824..0.64)

COMPRESION ARITMETICA

Idea: Convertir la probabilidad en un cierto número de bits.

Ej: **10110** (supongamos 1=⅗, 0=⅖)

[0.5824..0.60544) = 0 [0.60544..0.64) = 1

=> Resultado = [0.5824..0.60544)

COMPRESION ARITMETICA

.5824 = 0.10010
.60544= 0.10011

=> 1001 (2^-1 + 2^-5 + 2^-5) = 0.59375

FIN