Estudio sobre el desempeño del Algoritmo de Dispersión de Información

Betzayda D. Velázquez Méndez
Posgrado en Ciencias y Tecnologías de la Información, UAM-Iztapalapa
Betza.uam@gmail.com

Ricardo Marcelín Jiménez
Depto de Ingeniería Eléctrica, UAM-Iztapalapa
calu@xanum.uam.mx

Resumen

El algoritmo de dispersión de información (IDA por sus siglas en inglés), es un componente relevante en varios sistemas de almacenamiento de altas prestaciones. En sus aspectos fundamentales se trata de un conjunto de productos matriz - vector, realizados sobre un campo finito. En este trabajo mostramos el impacto del orden del campo sobre la velocidad del algoritmo. En particular, cuando se realizan sobre campos de característica 2, que pueden entenderse como cadenas de dígitos binarios. En un primer caso, se trabajan con cadenas de 8 bits, y en un segundo caso, con cadenas de 16 bits. Esta diferencia en longitud, tiene un impacto significativo sobre las operaciones de lectura y escritura de archivos.

Palabras Claves: Algoritmo de Dispersión de Información, Almacenamiento de información, Campos de Galois.

Abstract

The Information Dispersal Algorithm (IDA), is a relevant component in several high-performance storage systems. In its fundamental aspects it is a set of matrix - vector products, over a finite field. In this paper we show the impact of field order on the speed of the algorithm. In particular, when they are performed on fields with characteristic equal to 2, which can be understood as strings of binary digits. In a first case, we work with 8-bit strings, and in a second case, with 16-bit strings. This difference in length has a significant impact on the operations of reading and writing files.

Keywords: *Galois Field, Information Dispersal Algorithm, Information Storage.*

1. Introducción

La información se ha convertido en un activo estratégico de las organizaciones, cuyo volumen y tasas de crecimiento obligan a revisar los mecanismos para su gestión. Entendida esta última como las herramientas para su almacenamiento, consulta, transporte e incluso, generación de conocimiento.

Si se trata del almacenamiento de la información, es indispensable pensar en 2 requerimientos básicos que deben cuidarse en los sistemas diseñados para tal propósito: la alta disponibilidad y la escalabilidad. La alta disponibilidad se

DOI: 10.24275/uam/lerma/repinst/ciai2018/000171/Marcelin

consigue garantizando que el sistema prestará servicio aun si varios de sus componentes salieran de operación. En este sentido, la alta disponibilidad va de la mano de la confiabilidad y la tolerancia a fallas. Por cuanto se refiere a la escalabilidad, esta propiedad garantiza que un sistema puede crecer en sus capacidades sin perder calidad en el soporte de los servicios que ofrece.

La disponibilidad se consigue utilizando recursos redundantes. Existen 3 tipos de redundancia: de información, de equipos o física y de tiempo. La primera se refiere a codificar la información con un exceso de dígitos binarios tales que, si algunos de estos dígitos se perdieran, sería posible recuperar el mensaje original con los dígitos restantes.

La redundancia física se refiere a utilizar un conjunto de equipos de respaldo que podrían asumir las funciones de algún equipo caído en falla. Finalmente, la redundancia en tiempo se refiere a repetir una operación o procedimiento hasta tener la certeza que se ha completado con éxito o que debe abortarse.

La redundancia de la información se puede aplicar, ya sea para transmitir un mensaje o para almacenarlo. En este último caso, el término mensaje puede cambiarse o usarse como sinónimo de archivo. Alternativamente, se puede entender a un archivo como una secuencia finita de mensajes que deben almacenarse.

Bajo esta perspectiva, la forma más sencilla de redundancia de información consiste en guardar varias copias de un mismo archivo, sobre diferentes medios, por ejemplo discos o memorias de estado sólido. Sin embargo, esta solución no es la más eficiente, aunque si la más sencilla.

Un criterio fundamental para evaluar la eficacia de las técnicas empleadas para generar información redundante es la relación entre el factor de estiramiento (stretching) comparado contra la cantidad de fallas que pueden tolerarse. Por ejemplo, si tomamos un archivo y obtenemos 2 copias del mismo, de tal suerte que guardemos el original en un disco y cada copia en otros tantos discos, entonces se dice que esta solución tiene un factor de estiramiento del 200% y puede tolerar las fallas de 2 dispositivos, porque existirá un tercer dispositivo del que pueda recuperarse el archivo que se busca resguardar.

En el algoritmo de dispersión de la información (IDA), se tiene un archivo F, que consta de |F| dígitos binarios, este archivo se transforma en n archivos, llamados “dispersos” cada uno de los cuales es de tamaño |F|/m, donde $1 < m < n$, y tales que bastan cualesquiera m dispersos para recuperar a F. Se puede observar en este caso que el factor de estiramiento será (n/m) – 1 y las fallas que pueden tolerarse serán n-m.

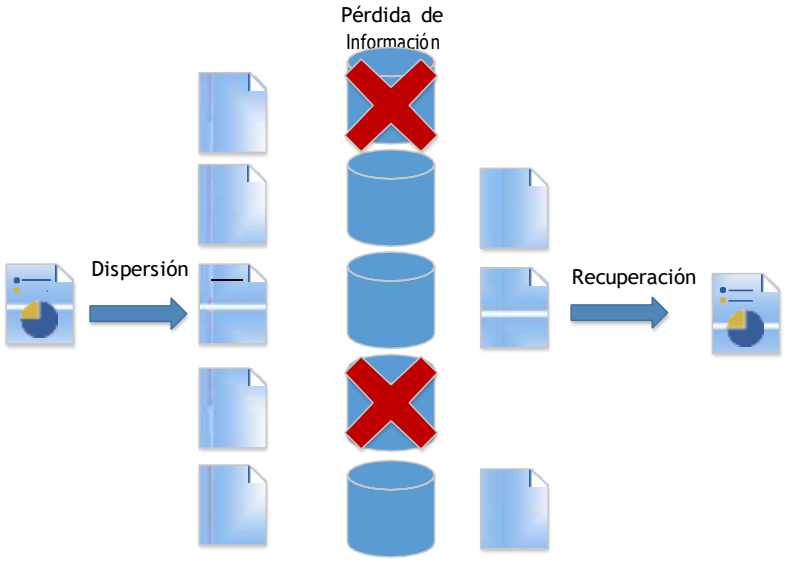

Figura 1. Funcionamiento de IDA

Dicho de otra forma, IDA es una solución parametrizable que permite encontrar diferentes puntos de equilibrio (trade-offs) entre la redundancia versus la tolerancia a fallas. Por ejemplo, en un IDA (5,3) se transformaría al archivo F en 5 dispersos, cada uno de ⅓ de |F| y tales que bastarían cualesquiera 3 de ellos para recuperar a F. El estiramiento que se conseguiría en este caso sería del 66% y con una tolerancia de 2 fallas. Vale la pena contrastar esta solución con la solución presentada en el párrafo anterior. De inmediato salta a la vista que IDA es superior, porque ofrece la misma tolerancia a fallas con un costo menor.

En 1990 Michael O. Rabin presenta posibles aplicaciones para IDA [1], desde entonces el algoritmo ha sido estudiado y utilizado para aplicaciones como el encaminamiento en redes de computadoras en paralelo tolerante a fallas [2], almacenamiento en la nube [3] e incluso Afianian utiliza IDA para el procesamiento de archivos en dispositivos móviles [4].

En la medida en que se ha reconocido la importancia económica de IDA, también se ha estudiado su costo de operación. En la base del algoritmo se encuentra una transformación lineal sobre un campo finito. En este trabajo presentamos un estudio en el que se evalúa la velocidad de esta operación sobre dos campos finitos de característica 2, denotados como $GF(2^8)$ y $GF(2^{16})$. Se concluye que el orden del campo tiene un impacto sensible en la velocidad con la que se procesa un archivo.

El resto de este artículo incluye las siguientes secciones: Métodos, donde se realiza una breve descripción del funcionamiento de IDA, además de la hipótesis del trabajo presentado; Resultados, se evidencia los valores obtenidos de IDA, así como las características en las que fue sometido; Discusión, se analiza la

congruencia de los resultados obtenidos con la hipótesis planteada; Conclusiones, se interpreta los resultados principales del trabajo presentado.

2. Métodos

Hay dos procesos fundamentales para IDA, la dispersión y la recuperación [5]. En el proceso de dispersión se utiliza una matriz de transformación A de n renglones por m columnas. Su requisito de construcción es que cualesquiera m de sus renglones formen un conjunto de vectores linealmente independientes y tal que cualesquiera m de sus renglones forman una submatriz cuadrada que es invertible.

Se considera que el archivo F está formado por una sucesión de vectores $\overrightarrow{b_1}, \overrightarrow{b_2}, \cdots \overrightarrow{b_N}$, sobre un campo finito arbitrario. Cada vector consta de m componentes i.e. tiene m dimensiones. Mediante una transformación lineal, cada vector $\overrightarrow{b_l}$, $l = 1, 2, \cdots, N$, se transforma en un vector $\overrightarrow{c_l}$ de n dimensiones, esto es

$$A \cdot \overrightarrow{b_l} = \overrightarrow{c_l}. \tag{1}$$

Supóngase ahora que, del vector $\overrightarrow{c_l}$ se pierden k de sus coordenadas, o bien, que se seleccionan m de sus componentes, en posiciones arbitrarias pero conocidas para formar el vector $\overrightarrow{d_l}$. Esto es equivalente a suponer que $\overrightarrow{d_l}$ se obtiene de a través de la transformación lineal

$$B \cdot \overrightarrow{b_l} = \overrightarrow{c_l}, \tag{2}$$

Donde a su vez B se construye seleccionando los m renglones de A en las mismas posiciones de las componentes de $\overrightarrow{c_l}$ que participan en $\overrightarrow{d_l}$. Por el requisito de la construcción, los m renglones de B son linealmente independientes y, por tanto, es invertible. En consecuencia, $\overrightarrow{b_l}$ puede reconstruirse a partir de $\overrightarrow{d_l}$ y B:

$$\overrightarrow{b_l} = B^{-1} \cdot \overrightarrow{d_l}. \tag{3}$$

Para esta implementación se ha utilizado los campos GF(2^8) y GF(2^{16}). El campo GF(2^8) contiene a todas las cadenas de 8 dígitos binarios; para esta implementación se ha generado a partir de su polinomio primitivo:

$$g(x) = x^8 + x^6 + x^5 + x^4 + 1. \tag{4}$$

En tanto, el campo GF(2^{16}) contiene todas las cadenas de 16 dígitos binarios y su polinomio primitivo es:

$$g(x) = x^{16} + x^{12} + x^3 + x + 1. \tag{5}$$

Tabla 1. Tiempos de dispersión y recuperación para archivos de diferentes tamaños

GF		1MB		4MB		16MB		64MB	
	n,m	Dis(s)	Rec (s)	Dis (s)	Rec (s)	Dis (s)	Rec (s)	Dis (s)	Rec (s)
8	5,3	0.078± 0.001	0.052± 0.0008	0.299± 0.011	0.204± 0.001	1.214± 0.037	0.815± 0.003	4.961± 0.088	3.360± 0.164
16		0.045± 0.003	0.030± 0.001	0.169± 0.004	0.117± 0.002	0.706± 0.058	0.477± 0.039	1.214± 0.099	0.815± 0.322
8	7,5	0.090± 0.002	0.072± 0.001	0.338± 0.002	0.278± 0.011	1.376± 0.028	1.119± 0.047	5.465± 0.092	4.412± 0.061
16		0.051± 0.001	0.042± 0.001	0.195± 0.006	0.162± 0.003	0.782± 0.026	0.649± 0.029	3.189± 0.110	2.634± 0.163
8	10,7	0.113± 0.002	0.089± 0.001	0.438± 0.015	0.349± 0.009	1.783± 0.067	1.396± 0.02	7.109± 0.118	5.656± 0.214
16		0.089± 0.0008	0.053± 0.002	0.248± 0.003	0.209± 0.017	1.013± 0.019	0.838± 0.052	4.076± 0.065	3.263± 0.082

Enseguida se construyen sus tablas de logaritmos y antilogaritmos, de tal modo que el tiempo de acceso a cualquier entrada de las tablas es constante y equivale al tiempo de acceso a los elementos de un vector de tamaño fijo. Ello implica que cualquier operación aritmética es de tiempo constante también.

La hipótesis de este trabajo consiste en asumir que el orden del campo, es decir, el total de sus elementos, incide en la velocidad con la que se realizan las operaciones directa e inversa del IDA.

3. Resultados

En el presente trabajo se desarrolló una implementación de IDA en lenguaje C sobre un sistema operativo Linux con un procesador Intel Core i7 y memoria RAM de 8GB. El compilador utilizado fue gcc versión 5.4.0. Los parámetros utilizados para realizar las respectivas pruebas de velocidad fueron las combinaciones (5,3), (7,5), (10,7). Para cada combinación se realizó la dispersión y recuperación en 4 archivos de tamaño 1MB, 4MB, 16MB y 64MB.

Asimismo, cada combinación fue ejecutada sobre los campos GF(2^8) y GF(2^{16}). Cabe mencionar que las ejecuciones sobre las combinaciones se repitieron 20 veces en cada campo. La Tabla 1 muestra el promedio y la desviación estándar, medidos en segundos, para cada conjunto de experimentos.

En la Figura 2 representamos de forma gráfica los tiempos obtenidos en segundos para el proceso de dispersión y recuperación agrupados para cada archivo, sobre el campo de $GF(2^8)$ y $GF(2^{16})$.

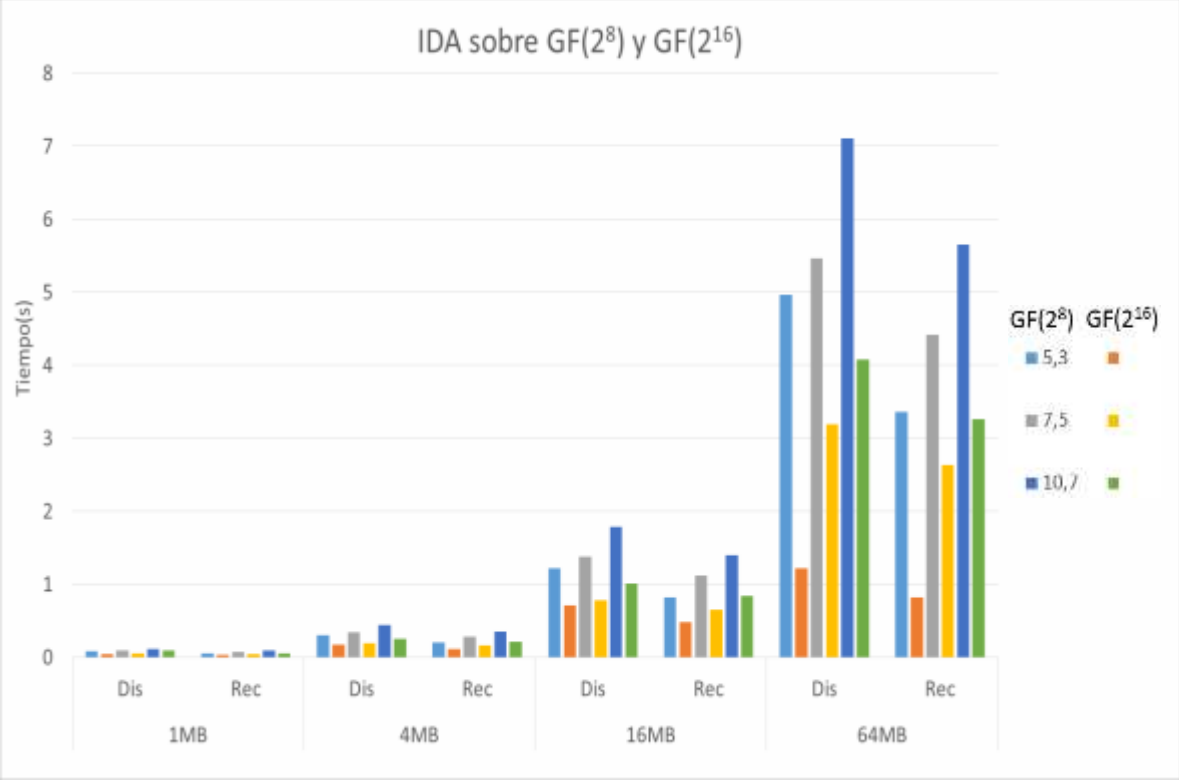

Figura 2. Tiempo de dispersión y recuperación sobre $GF(2^8)$ y $GF(2^{16})$.

4. Discusión

El tiempo de procesamiento depende directamente del tamaño del archivo. Asimismo, sobre archivos de mayor tamaño se puede observar mejor el beneficio de usar un campo finito de mayor orden. Puede verificarse en la Figura 2, por ejemplo, que para el archivo de 1 MB, los tiempos de dispersión y recuperación son muy pequeños, y no hay una diferencia notable en el uso de un campo u otro. En contraste, se observa una diferencia importante en estas operaciones, para el caso del archivo de 64MB.

Se puede apreciar que para cualquier instancia de IDA (5,3), (7,5), (10,7), la dispersión toma más tiempo que la recuperación, debido a que hay n archivos para producir, en cambio, en la recuperación sólo se trabajan m archivos, lo cual disminuye el tiempo de procesamiento. Queda pendiente estudiar el impacto de las operaciones de acceso a disco en el costo total del algoritmo.

El tiempo de ejecución de los algoritmos de dispersión y recuperación depende directamente de los parámetros m y n. Por otra parte, el tiempo de ejecución de los algoritmos de dispersión y recuperación depende inversamente del orden del campo con el que se trabaja.

Este último resultado puede explicarse, para los casos considerados, porque en el campo GF(2^8), todos los archivos implicados (fuente y dispersos) se leen y escriben en unidades de 8 bits de longitud. En tanto, en el campo GF(2^{16}) todas las operaciones se realizan sobre unidades de 16 bits de longitud. En este último caso, por ejemplo, el archivo fuente que se dispersa se lee en la mitad del tiempo.

5. Conclusiones

- Utilizar un campo de mayor orden, reduce el tiempo de procesamiento. Al menos para los casos considerados.
- Es preferible utilizar un campo GF(2^{16}) con archivos de mayor tamaño, de lo contrario, no tendría una ganancia significativa de velocidad.
- El tiempo de dispersión será mayor al tiempo de recuperación de un archivo
- De igual forma, los parámetros de IDA permiten encontrar diferentes compromisos (trade-offs) entre la redundancia y la tolerancia a fallas.

6. Bibliografía y Referencias

[1] Rabin M. O., The Information Dispersal Algorithm and its Applications, Springer, New York, 1990.

[2] Yuh-Dauh L., Information Dispersal and Parallel Computation, Cambridge: Cambridge University Press, 1993.

[3] Lee O. T., Kumar M. SD., Chandran P., Erasure coded storage systems for cloud storage—challenges and opportunities, Data Science and Engineering (ICDSE), 2016 International Conference on, 2016, pp. 1-7.

[4] Afianian A. Nobakht S. S., Ghaznavi-Ghoushchi M. B., Energy-efficient secure distributed storage in mobile cloud computing, 2015 23rd Iranian Conference on Electrical Engineering, 2015, pp. 740-745.

[5] Rabin M. O., Efficient dispersal of information for security, load balancing, and fault tolerance, ACM Digital Library, vol. 36, nº 2, pp. 335-348, 1989.